ROUTING AND SCHEDULING FOR CLOUD SERVICE DATA CENTERS

BY

QIAOMIN XIE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Assistant Professor Yi Lu

ABSTRACT

In recent years, an increasing variety of dynamic-content web services, such as search, social networking and on-line commerce, have been moved to the Cloud service data centers. One of the features offered by Cloud data centers is on-demand service. To achieve this, current Cloud data centers are designed with an excess of provision for highly dynamic work load. Problems with such design include low server throughput and lack of scalability, which are considered as very important challenges for attaining system efficiency. This research aims to develop novel algorithms for the Cloud data center to achieve good performance while maintaining cost and energy efficiency.

In general, Cloud service data centers consist of the front-end and back-end systems. To ensure a good level of service performance, neither the front-end nor back-end system should be neglected at the design of a Cloud data center. This study investigates features and challenges for Cloud service data centers. For the front-end system, the distributed design of load balancers is highly favored for achieving scalability. A novel algorithm is proposed for large-scale load balancing with distributed dispatchers. Both analysis and simulation show the advantage of the proposed algorithm over the state of the art.

In the back-end system, intensive data processing is required to search, analyze and mine the vast amount of data. Cluster computing systems, like MapReduce and Hadoop, have provided an efficient platform for large scale computation. This research studies the data locality problem for cluster computing systems, which significantly affects system throughput and job completion time. In particular, a new task assignment algorithm is proposed and shown to significantly outperform the current implementation.

ACKNOWLEDGMENTS

I wish to express my deepest gratitude to my advisor, Prof. Yi Lu, for her guidance and omnipresent support along the way. Her passion for research and her intense commitment to her work inspire me a lot. It is a pleasure to work with her.

I appreciate the great support from the staff at the Coordinate Science Lab. Thank you Barbara J. Horner, Wendy Kunde and Carol Wisniewski. Thanks also to my fellow graduate students for making my free time more enjoyable.

I also would like to thank the Air Force Research Laboratory (AFRL) and Air Force Office of Scientific Research (AFOSR). This material is based on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084.

Lastly but not the least, it is my greatest honor to thank my family for their love and support.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS v

CHAPTER 1 INTRODUCTION 1
1.1 Architecture of the Cloud 2
1.2 Challenges 3
1.3 Organization of this Thesis 6

CHAPTER 2 FRONT-END LOAD BALANCING 7
2.1 Previous Work 7
2.2 The Join-Idle-Queue Algorithm 9

CHAPTER 3 ANALYSIS OF THE JIQ ALGORITHM AND DISCUSSION 12
3.1 Main Analytical Results 12
3.2 Evaluation via Simulation 19
3.3 Extension of JIQ 19

CHAPTER 4 BACK-END TASK ASSIGNMENT WITH DATA LOCALITY CONSTRAINT 24
4.1 Discrete-Time Model 24
4.2 Algorithm Description 25
4.3 Mean-Field Model for a Single Time Slot 26
4.4 Performance for a Single Time Slot 29
4.5 Fixed Point Characterization 33
4.6 Performance in Continuous Time Model 37

CHAPTER 5 CONCLUSION 39

APPENDIX A PROOF OF THEOREM 3 41

APPENDIX B EVOLUTION OF RESIDUAL GRAPH WITH TASKS QUEUEING 45

REFERENCES 52

LIST OF ABBREVIATIONS

LB	Load Balancer
PS	Processor Sharing
FIFO	First-In-First-Out
JSQ	Join-Shortest-Queue
JIQ	Join-Idle-Queue

CHAPTER 1

INTRODUCTION

The increasing popularity of web services, such as search, social networking and on-line commerce, has accelerated a trend toward server side computing. While early web services are mostly concerned with simple information retrieval, today web applications involve complex computing. Another driving force is the explosive growth of data. Today petabytes of data are being collected, processed and analyzed. These data sets are too large to be transmitted economically over the Internet. Data centers with large-scale storage and processing capacity are needed to store, manage, and process of this vast amount of data. With the shift of computing and data storage from desktop or local servers to large data centers, various web services are conveniently accessible to end users through lightweight desktop or mobile applications.

Cloud data centers have emerged as a popular computing platform for providing an increasing variety of web services. Operation efficiency and agility advantages, together with great economic benefits of the Cloud, have driven the migration of web applications and services to the Cloud [1]. Companies like Google, Microsoft, Yahoo and Amazon have built large data centers to provide web services.

For all web services, an important metric is the request response time, which determines the user satisfaction. Google and Microsoft Bing [2] agree that server delays have significant negative impacts on usage by users. Moreover, the cost of delay increases over time and persists. Similarly, it has been shown that every 100 ms increase in load time of Amazon.com decreased sales by 1% [3].

On the other hand, as the quality of services depends on the aggregate processing of a large platform, it also determines cost of data centers. One distinct feature offered by the Cloud data center is on-demand service. To achieve this, current Cloud data centers are designed with an excess of provision for highly dynamic work load. Problems with such design include

low server throughput and lack of scalability, which are considered as very important challenges for attaining system efficiency.

This research aims to explore software development for the Cloud service data center to achieve high performance. Many mechanisms have been exploited for traditional web server farms to minimize response time. However, the existing approaches fall short with the large scale and distinct structure of service-oriented data centers.

1.1 Architecture of the Cloud

Multi-tier architecture is a common design for Cloud data centers. Typically, the three-tiered model is widely used, which includes web-server, application and database tiers.

- The web-server tier provides the clients access to the data center. Servers in this tier are involved with forwarding the requests from clients to the back end system for processing, assembling results and presenting the response to the client.
- The application and database tiers work together to perform the detailed processing for incoming requests and send the results to the web-server tier.

For a Cloud data center, its front-end system refers to the web-server tier, and its back-end system consists of the application and database tiers.

Compared with traditional web server farms, Cloud service data centers have the following distinct features:

- Large scale of the front end: Cloud have hundreds or thousands of servers for the front end alone, while a traditional web server farm contains only a few servers.
- Complex computation at the back end: due to the explosive growth of data stored at the Cloud data center, requests processed at the back end require a large amount of computation to extract the information. In addition, today's service requests are becoming more complex, which also contributes to the increasing complexity of computation.

In contrast, most traditional web services involve simple page fetching without much computation.

1.2 Challenges

Improving the overall performance of the Cloud web service requires a great deal of research effort for both the front-end and back-end systems.

1.2.1 Load balancing at the front end

Load balancing has been well developed for traditional web server farms to improve performance. It achieves short response time by distributing load across multiple servers. In traditional small web server farms, a centralized hardware load balancer, such as F5 Application Delivery Controller [4], is used to dispatch jobs evenly to the front-end servers. The problem of routing policy is extensively studied [5, 6, 7]. Among the various centralized algorithms, the `Join-the-Shortest-Queue` (JSQ) is the most popular algorithm used in server farms with Processor Sharing (PS) discipline. With `JSQ` algorithm, the balancer dispatches an arriving request to the server with the least number of unfinished jobs. From the view point of an incoming job, the `JSQ` algorithm is a greedy algorithm, as the algorithm grants the job the highest instantaneous processing speed. The `JSQ` is favored for its near-insensitivity for all job size distributions and near-optimality [8]. Another benefit of JSQ is its low communication overhead in traditional web server farms, as the load balancer can easily track the number of outstanding jobs at each server with the information of all arrivals and departures at this particular server. Hence the `JSQ` algorithm does not incur extra communication.

Two distinct features of the Cloud make the existing load balancing mechanisms for traditional web server farms inefficient in the Cloud environment.

1. Scalability. As one of the major advantages of the Cloud, scalability, or elasticity, makes it appear with infinite computing resources available on demand [9]. It is reported that data centers have an average server utilization rate ranging from 5% to 20% [10]. However, for many services the peak workload reaches up to 2-10 times of the average load. To achieve service provision and cost efficiency, the Cloud data centers should enable horizontal scaling,

i.e. adding (turning off) servers to accommodate increasing (decreasing) demand. In addition to server scalability, dynamically adding load balancers to distribute load among servers is also an important point in achieving scalability. For a system with a central hardware load balancer, it is difficult to accomplish reliability and scalability. The failure of the single load balancer would result in death of the whole system. In case of servers scale-out, the single load balancer would be overloaded. Cost is another drawback of implementing hardware load. These drawbacks have motivated the development of distributed software load balancers in the Cloud [11].

Figure 1.1 illustrates load balancing with distributed dispatchers. Requests are routed to a random dispatcher via, for instance, the Equal-Cost-Multi-Path (ECMP) algorithm in a router. Load balancing of flows across dispatchers is less of a problem as the numbers of packets in web requests are similar. The service time of each request, however, varies much more widely, as some requests require the processing of a larger amount of data. Each dispatcher independently attempts to balance its jobs. Since only a fraction of jobs go through a particular dispatcher, the dispatcher has no knowledge of the current number of jobs in each server, which makes the implementation of the `JSQ` algorithm difficult.

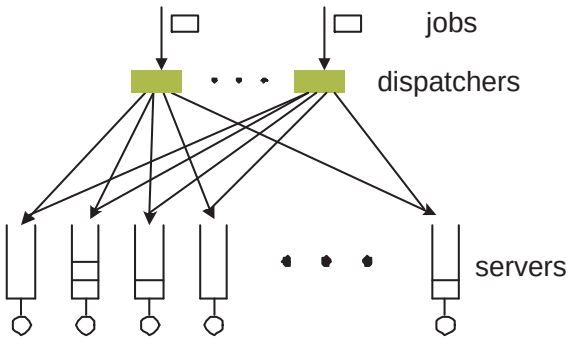


Figure 1.1: Distributed dispatchers for a cluster of parallel servers.

2. Large scale front-end. Another main distinction between traditional web server farms and Cloud data centers lies in the scale of front-end servers. It is common that a Cloud data center contains thousands of servers. For centralized load balancing algorithms like `JSQ`, each dispatcher needs to have knowledge of each server's state, which incurs high communication overhead for a system with distributed dispatchers.

In this research we developed a new algorithm for distributed load balancing in a large system.

1.2.2 Data locality at the back end

The challenges of the Cloud service back-end system rise from the massive data collection, which is too large to be transmitted economically over the Internet. Today's web service involves intense data computing that produces small data output. The notion of parallel processing has been put forward to achieve efficient large-scale computation. Cluster computing systems, such as MapReduce [12], Hadoop [13] and Dryad [14], have become a prevalent distributed computing platform for data-intensive applications.

In cluster computing systems, a file is divided into many small chunks to facilitate parallel computation. Moreover, each data chunk has several replicas to ensure data availability in the event of failure [15, 16, 17]. By default, each data chunk is replicated three times and placed on different servers: The first chunk is placed on a randomly selected server; the second chunk is placed on a random server in the same rack, in order to protect against server failures; and the third chunk is placed on a random server outside the rack to ensure data availability in case of rack failures. When a task is placed on a server where data are not locally available, it will need to retrieve data from one of the remote servers hosting the replicas. This increases the completion time of the task. As a result, placing tasks as close as possible to data is a common practice of data-intensive systems, referred to as the *data locality* problem [12, 14, 18].

Data locality is an important problem as it significantly affects system throughput and job completion times. The current scheduling algorithms in Hadoop are based on the Random Server algorithm [19], which depends on a large number of outstanding tasks to achieve high data locality. As a result, with light to medium load, which is the region online systems operate in today, the Random Server algorithm results in unnecessary delay of tasks.

In this research, we propose a degree-guided task assignment algorithm that significantly outperforms the Random Server algorithm at light to medium load.

1.3 Organization of this Thesis

The objective of this research is to achieve fast response time and high-throughput processing of the Cloud service data center by exploring efficient algorithms for both the front-end and back-end system. This thesis describes the design and analysis of proposed algorithms. It is organized as follows:

Chapter 2 starts with a review of previous work in load balancing and then presents our novel algorithm, `Join-Idle-Queue(JIQ)`, for systems with distributed dispatchers.

Chapter 3 presents the analysis and evaluation of the `JIQ` algorithm and discusses the extension of the `JIQ` algorithm for various cases.

Chapter 4 presents a degree-guided back-end task assignment algorithm with data locality constraint. It includes the analysis of this algorithm and characterization of its performance.

Chapter 5 concludes the thesis.

CHAPTER 2

FRONT-END LOAD BALANCING

In this chapter, we focus on load balancing on the web-facing front end of Cloud service data centers.

2.1 Previous Work

We present both technologies used in industry and related theoretical literature in this section.

2.1.1 Techniques in industry

In general, the details of data center design for enterprises are not available to the public. This prevents us from exploring comprehensive techniques used for load balancing in existing cloud clusters. For some cloud platforms, we only know the type of load balancers (software or hardware) they are based on without details of the policies employed inside.

Hardware-based load balancer (LB), like F5 BIG-IP, is used widely in Cloud systems, such as Microsoft Azure [20] and GoGrid [21]. The application specific hardware components in hardware LB endow it with powerful processing, while also make it much more expensive than software LB, such as Nginx [22] and HaProxy [23]. Cloud clusters based on software LB are also prevalent, like Amazon EC2 [24] and OpenNebula [25].

Here we summarize several algorithms that are currently in wide use in hardware and software load balancers.

1. `Join-the-Shortest-Queue (JSQ)`: `JSQ` is also referred as 'Least Connections'. The cost of near-optimality of JSQ is to track of all servers' states, which incurs extensive communication overhead.

2. `Round-Robin (RR)`: All servers are kept in a circular queue. The central dispatcher goes around this queue and directs an arriving request to one server at a time. It accomplishes equal load distribution across all servers. Round-Robin can be easily implemented, but has poor performance compared with `JSQ`. Moreover, its performance deteriorates with heterogeneous servers.
3. `Weighted-Round-Robin (WRR)`: Instead of distributing load evenly among all servers, the number of requests assigned to a server is proportional to a pre-defined weight for this server. The `WRR` produces improved performance over `RR` for heterogeneous systems.

2.1.2 Related theoretical literature

The main challenge for distributed implementation of systems with large-scale front-end servers is to avoid intensive communications between dispatchers and servers. For the Cloud, algorithms with little or no states are preferred. A simple solution is the randomized algorithm, where an incoming request is dispatched to a randomly sampled server. Randomized algorithm is easy to implement and does not require extra communication. Nevertheless, the resulting large average response time restrains its application. The following approaches might be adaptable for this problem, as they are based on randomized algorithm and get rid of overwhelming communication.

The Power-of-d (SQ(d)) algorithm

Under the `SQ(d)` algorithm, at the arrival of a request, $d \geq 2$ servers are sampled randomly and the one with the least number of jobs is selected to process this request. There has been a large amount of work on the theoretical analysis of the `SQ(d)` algorithm [26, 27, 28, 29, 30]. Compared with the Randomized algorithm and the `JSQ` algorithm, the `SQ(d)` algorithm makes a trade-off between performance and communication overhead. Note that communication between the dispatchers and servers is still required for the `SQ(d)` algorithm upon request arrival. This increases the overall response time. In addition, the performance of `SQ(d)` is far from optimal. Hence the `SQ(d)` algorithm might not be efficient in a Cloud service data center.

Work stealing and load sharing

As an important technique to achieve load balancing for shared-memory

multiprocessor systems, work stealing has been extensively studied [31, 32, 33]. Under work stealing, idle processors attempt to steal work from randomly sampled processors. Another technique, load sharing, is developed for distributed systems [34, 35, 36]. Contrary to the initiative of the underutilized processors, in load sharing, it is the overloaded processors that migrate work to underutilized processors. These two approaches have been proven to be efficient for many circumstances. However, directly applying work stealing/sharing to the Cloud service cluster is not appropriate. Shared-memory multiprocessors have little difficulty pulling or pushing a job that has already started execution, while for the Cloud cluster, it involves in tedious migration of TCP connection, leading to increase of response time. In addition, moving jobs between servers in the Cloud incurs extra communication overhead.

2.2 The Join-Idle-Queue Algorithm

We propose a class of novel algorithm, `Join-Idle-Queue` (JIQ), for the Cloud data center with distributed dispatchers [37]. The main idea is to remove the discovery of lightly loaded servers from the critical path of requests assignment. Under the `JIQ` algorithm, lightly loaded servers themselves inform the dispatchers their states, rather than being discovered by dispatchers via sampling. This strategy eliminates the communication between dispatchers and servers at the arrival of requests, leading to shorter response time for requests. Moreover, the reporting communication from servers to job dispatchers is much less costly as it can ride on the heartbeats that inform dispatchers of the health state of servers.

2.2.1 Motivation

The idea of exploiting idle servers is inspired by the essence of load balancing. Observe that an efficient load balancing changes the arrival rate for a server based on its state: it increases the arrival rate to idle servers while decreasing the arrival rate to busy servers. Then for a single server, its busy cycles will be shorter, which indicates faster response time.

To have a better understanding of the effect of load balancing schemes on arrival rates, compare the rate of arrival to an idle processor, λ_0, for the

following three algorithms. Consider a system with rate-$n\lambda$ arrivals and n processors of service rate 1; then the system load is λ.

$$\begin{aligned}
\texttt{Random}. \quad & \lambda_{0,R_1} && = \lambda, \quad \forall\ n. \\
\texttt{SQ(d)}. \quad & \lim_{n\to\infty} \lambda_{0,R_d} && = \lambda \frac{1-\lambda^d}{1-\lambda} = \lambda + \lambda^2 + \cdots + \lambda^d. \text{ [28]}. \\
\texttt{JSQ}. \quad & \lim_{n\to\infty} \lambda_{0,JSQ} && = \frac{\lambda}{1-\lambda} = \lambda + \lambda^2 + \lambda^3 + \cdots \text{ [8]}.
\end{aligned}$$

In [38], the `Random` algorithm is shown to result in the independent stochastic queueing process at each processor. The arrival rate is constant for all queue sizes. Under the `SQ(d)` algorithm, one term is added to the arrival rate with an increase in d, which is the number of processors it compares. Comparing the `JSQ` with the `SQ(d)` algorithm, we have an interesting observation,

Corollary 1

$$\lim_{d\to\infty} \lim_{n\to\infty} \lambda_{0,R_d} = \lim_{n\to\infty} \lambda_{0,JSQ}.$$

Note that among these three algorithms, the `Random` algorithm produces the worst response times and the `JSQ` algorithm achieves the best performance. Under `JSQ`, the queue sizes never exceed 1 in the large system limit, i.e., n goes to infinity. Every job is hence directed to an idle server. This motivates our focus on idle servers.

2.2.2 Algorithm description

We introduce a new data structure, called *I-queue*, for the `JIQ` algorithm. In the basic version, an I-queue contains a list of idle servers that have informed this dispatcher at the time of their idleness. Each dispatcher has an I-queue. The overall system of the `JIQ` algorithm is shown in Fig. 2.1.

Consider a system consisting of n homogeneous servers in parallel and m dispatchers, with $m << n$. We assume that the dispatcher can assess all servers. As illustrated in Fig. 2.1, each dispatcher is attached with an I-queue, which contains a subset of idle servers at the system. The dispatchers strive to balance the load by assigning requests to idle servers, which are

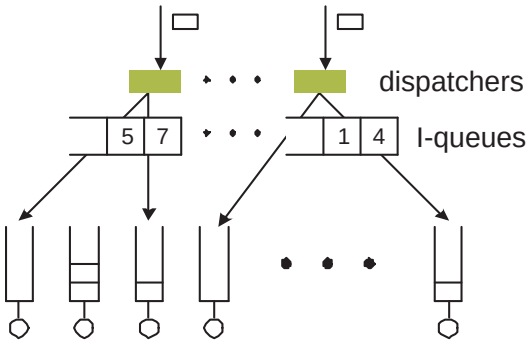


Figure 2.1: The JIQ algorithm with distributed dispatchers, each of which has an I-queue attached.

present at the I-queue. The distributed implementation raises the problem of how to distribute idle servers evenly at the dispatchers. To solve this problem, the `JIQ` algorithm balances the idle servers in the reverse direction, which is called the secondary load balancing. Note that the performance of the primary load balancing for assigning requests to servers highly relies on the secondary load balancing. These two systems communicate with the I-queues.

Primary load balancing: The primary load balancing system assigns arriving requests to servers based on the information of idle servers present in the I-queues. When a request arrives, the dispatcher first checks the state of the attached I-queue. If the I-queue is empty, the request is assigned to a randomly sampled server. And in this report, such requests are called random arrivals for servers. Otherwise, the first idle server at the I-queue is removed to process this request.

Secondary load balancing: The secondary load balancing system is designed to balance idle servers across dispatchers. Under the current version of algorithm, an idle server informs only one dispatcher of its idleness, i.e., joins the corresponding I-queue. We consider the following load balancing algorithms for the idle servers: `Random` and `SQ(d)`. Under the `Random` algorithm, a server joins a randomly selected I-queue when it becomes idle. With the `SQ(d)` algorithm, an idle server samples d I-queues, and joins the one with the least number of idle servers. We refer to the algorithm with `Random` load balancing in the reverse direction as `JIQ-Random` and that with `SQ(d)` as `JIQ-SQ(d)`.

CHAPTER 3

ANALYSIS OF THE JIQ ALGORITHM AND DISCUSSION

Using both the analytical study and simulation, we show the advantage of the `JIQ` algorithms over other existing algorithms.

3.1 Main Analytical Results

The web service infrastructure has evolved from small server farms to the Cloud data center, which shows a trend toward more wide use of large-scale systems. It is hence interesting to investigate algorithm performance in the large system limit, i.e., with hundreds or thousands of servers.

3.1.1 Model for analysis

Consider a system consisting of n homogeneous servers in parallel and m dispatchers, with $m << n$. In the large system limit, i.e., $n, m \to \infty$, while the ratio of the number of servers to the number of I-queues is fixed,

$$r \equiv \frac{n}{m}.$$

Requests arrive at the system as a rate-$n\lambda$ Poisson process, $\lambda < 1$, hence the load on the system is λ. Each request is directed to a randomly selected dispatcher, which assigns it to one server according to the `JIQ` rule. The service times of requests are drawn i.i.d. from a distribution $B(\cdot)$ with mean 1.

For simplicity, we make the following two assumptions: 1. Each idle server has exactly *one copy* in the I-queues. 2. *Only* idle servers are present in the I-queues.

3.1.2 Analysis of secondary load balancing system

In order to analyze the equilibrium of the secondary system, we need a better understanding of the arrival and departure processes at the I-queues. For the secondary load balancing system, its "servers," I-queues, receive arrivals of idle servers. Although the arrival process to I-queues is not Poisson in a finite system, we proved that it goes to Poisson as $n \to \infty$, with the `Random` or `SQ(d)` load balancing scheme in the secondary system [37].

Lemma 1 *Let the arrival process at I-queue* 1 *be* $\Lambda_n(t)$*. For* ***JIQ-Random****,* $\Lambda_n(t)$ *goes to a Poisson process as* n *goes to infinity. For* ***JIQ-SQ(2)****,* $\Lambda_n(t)$ *goes to a state-dependent Poisson process where the rate depends on the I-queue lengths.*

Here we are interested in the fraction of occupied I-queues, as it determines the rate of arrivals through occupied I-queues and arrivals that are randomly assigned. The two rates will determine the response time of the primary system. Note that the expected proportion of occupied I-queues is equal to the "load" on I-queues.

Note that the requests arrival process at each dispatcher is Poisson with rate $n\lambda/m$, which is exactly the "service" rate of each "server" at the secondary load balancing system. Hence the system of I-queues have exponential "service" times. For both the `Random` and `SQ(d)` algorithms, the exponential distribution of "service" time allows the establishment of an explicit relation between the virtual load on I-queues and the mean queue length.

On the other hand, it is the idle servers that build up the queues at I-queues. Observe that the expected proportion of idle servers goes to $1-\lambda$, in spite of the load on I-queues. Therefore the mean queue length of an I-queue goes to $r(1-\lambda)$.

Lemma 2 *Consider a M/G/n system with arrival rate* $n\lambda$*. Let the random variable* μ_n *denote the number of idle servers in equilibrium. Let* $q_n = \frac{\mu_n}{n}$*, then*

$$\mathbb{E}(|q_n - (1-\lambda)|) \to 0 \quad as \quad n \to \infty. \tag{3.1}$$

With the information above, we can characterize the proportion of occupied I-queues using the actual load on the system [37].

Theorem 1 Proportion of occupied I-queues. *Let ρ_n be the proportion of occupied I-queues in a system with n servers in equilibrium. We show that*

$$\mathbb{E}(|\rho_n - \rho|) \to 0 \quad as \quad n \to \infty, \tag{3.2}$$

where
for the ***JIQ-Random*** *algorithm,*

$$\frac{\rho}{1-\rho} = r(1-\lambda), \tag{3.3}$$

for the ***JIQ-SQ(d)*** *algorithm,*

$$\sum_{i=1}^{\infty} \rho^{\frac{d^i-1}{d-1}} = r(1-\lambda). \tag{3.4}$$

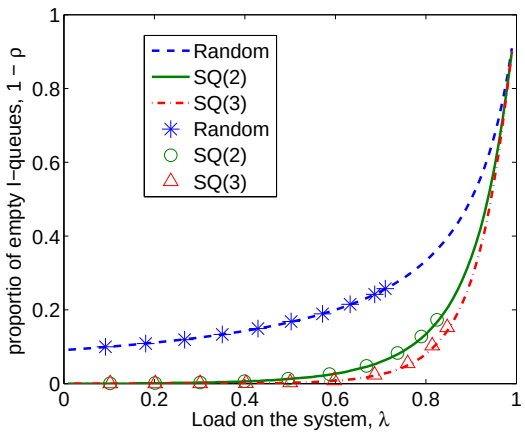


Figure 3.1: Proportion of empty I-queues with $r = 10$. Markers are from simulations with $n = 500$ and $m = 50$. No visible difference is observed for different service time distributions. Line curves are values computed by Theorem 1.

Figure 3.1 shows the proportion of empty I-queues, $1 - \rho$, with $r = 10$ for the `Random` and `SQ(d)` algorithms. Observe that simulation results for finite system with $n = 500$ and $m = 50$ fit well with values obtained from Theorem 1. In addition, the results confirm the invariance of the proportion of occupied I-queues with different jobs service time distributions.

The advantage of the `SQ(d)` algorithms over the `Random` algorithm is also verified. In particular, with low to medium load, the `SQ(d)` algorithms result

in much lower proportion of empty I-queues than the `Random` algorithm. And probing two I-queues seems sufficient, as the further reduction of the proportion of empty I-queues by `SQ(3)` over `SQ(2)` is not obvious. At high load, most I-queues are empty as the number of idle servers is small. A better strategy is needed in order to make use of the I-queues.

3.1.3 Analysis of primary load balancing system

The response time of the primary system can be derived with the results from the secondary load balancing system. Let

$$s = \lambda(1 - \rho),$$

where ρ is the proportion of occupied I-queues computed in Theorem 1. In [37], we proved that the primary system has the following properties .

Theorem 2 Queue Size Distribution. *Let the random variable Q_n denote the queue size of the servers of an n-system in equilibrium. Let Q_s denote the queue size of a M/G/1 server at the reduced load s with the same service time distribution and service discipline. Then*

$$\mathbb{P}(Q_n = k) \;\rightarrow\; \frac{\mathbb{P}(Q_s = k)}{1 - \rho} \;\; \forall\; k \geq 1 \;\; as \;\; n \rightarrow \infty. \tag{3.5}$$

Corollary 2 Mean Response Time. *Let $\bar{q}$ denote the mean queue size at the servers in the large system limit. Then*

$$\bar{q} \;=\; \frac{q_s}{1 - \rho}, \tag{3.6}$$

where q_s is the mean queue size of the M/G/1 server with the same service time distribution and service discipline.

The mean response time

$$\bar{T} = \frac{\bar{q}}{\lambda} = \frac{q_s}{s},$$

assuming a mean service time of 1.

Another important property of the `JIQ` algorithms is its insensitivity to various service time distributions.

Corollary 3 Insensitivity. *In the large system limit, the queue size distribution under the* `JIQ` *algorithm with PS service discipline depends on the service time distributions only through its mean.*

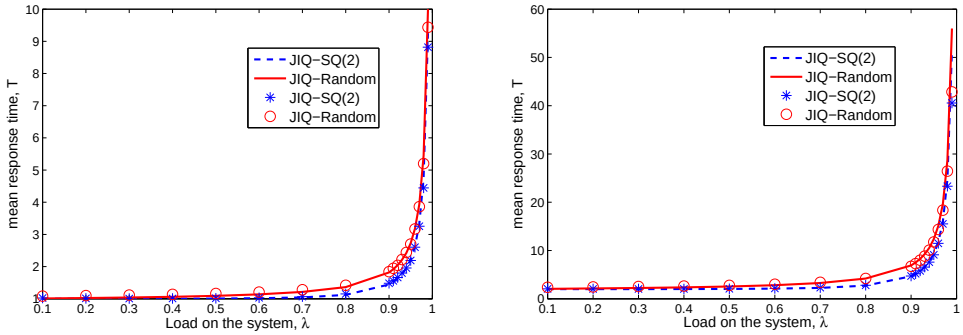


Figure 3.2: (a) Comparison of mean response time for the `JIQ-Random` and `JIQ-SQ(2)` algorithms with exponential service time distribution, which has mean 1 (this makes the minimum possible mean response time 1). (b) Comparison of mean response time for the `JIQ-Random` and `JIQ-SQ(2)` algorithms with a Weibull service time distribution, which has mean 2 and variance 20 (minimum possible mean response time is 2). Both figures are for system with $r = 10$. Markers are from simulations with $n = 500$ and $m = 50$. Line curves are obtained from Theorem 2.

Figure 3.2(a) shows the mean queue size for general service disciplines with exponential service time distributions and Fig. 3.2(b) shows the mean queue size for FIFO service discipline with Weibull service time distribution with mean 2 and variance 20. We observe that the simulation results for a finite system are consistent with the computed values from Corollary 2 for both cases. The insensitivity of the `JIQ` to different service time distributions is also verified here.

It is interesting to observe that the `JIQ-Random` and `JIQ-SQ(2)` algorithms result in similar mean response times. This can be explained as follows: With $r = 10$, random arrivals is too small to incur large queue sizes at low loads, and the improvement of `JIQ-SQ(2)` over `JIQ-Random` in ρ is not significant at high loads. The `JIQ-SQ(2)` algorithm is expected to show more obvious advantage over`JIQ-Random` with a big processor to I-queue ratio, r. We will show the effect of r on the performance of different algorithms in the next section.

3.1.4 Reduction of queueing overhead

With the analytical results for the `JIQ` algorithms, it is possible to characterize the effects of system parameters on the performance. In particular, we can compute the mean response time for `JIQ-Random` as Eqn. (3.3) provides an explicit expression for ρ. Here we consider both the PS and FIFO service discipline for the servers.

1. PS

For a $M/G/1$ queue with PS discipline, it is well known that the stationary distribution of the queue length is independent of the job service time distribution apart from its first moment [39]. Its mean response time is:

$$q_s = \frac{s}{1-s}, \tag{3.7}$$

where s is the load for the queue.

From Corollary 2, we can calculate the mean response time for `JIQ-Random`:

$$\bar{T}_{PS} = \frac{1}{1-s} = \frac{1}{1-\frac{\lambda}{1+r(1-\lambda)}} = 1 + \frac{\lambda}{(1-\lambda)(1+r)}.$$

Note that for the system with the `Random` algorithm, each server behaves as a M/G/1 queue with rate-λ arrival independently. The corresponding mean response time is:

$$\bar{T}^R_{PS} = \frac{1}{1-\lambda} = 1 + \frac{\lambda}{1-\lambda}.$$

As the mean service time of a job is 1, the queueing overhead is $\frac{\lambda}{(1-\lambda)(1+r)}$ for `JIQ-Random` and $\frac{\lambda}{1-\lambda}$ for `Random`. The `JIQ-Random` algorithm achieves a $(1+r)$-fold reduction over the textttRandom algorithm.

2. FIFO

Consider a $M/G/1$ queue with FIFO discipline. It is easy to compute the mean response time by using the Pollaczek-Khinchin formula,

$$q_s = s + s^2\frac{1+C^2}{2(1-s)}, \tag{3.8}$$

where $C^2 = \sigma^2/(\bar{x})^2$, the ratio of the variance of the particular service time distribution to its mean squared.

Similarly, we can obtain the mean response time for `JIQ-Random` with FIFO:

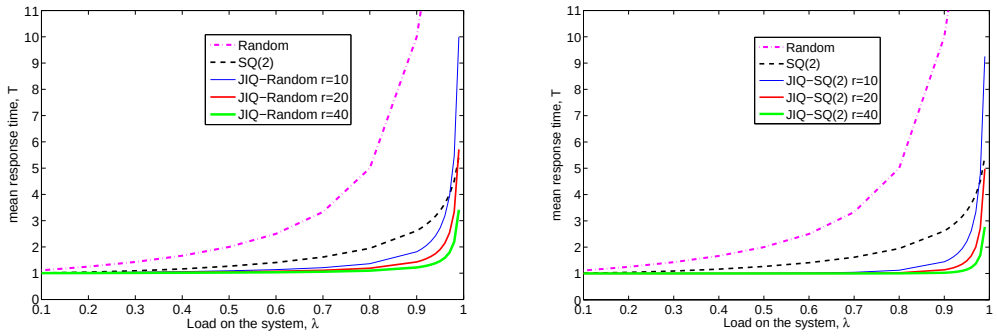
$$\bar{(T)}_{FIFO} = 1 + s\frac{1+C^2}{2(1-s)} = 1 + \frac{1+C^2}{2}\frac{\lambda}{(1-\lambda)(1+r)}.$$

Compare this to the mean response time of a M/G/1 queue with rate-λ arrival

$$\bar{T}^R_{FIFO} = 1 + \lambda\frac{1+C^2}{2(1-\lambda)} = 1 + \frac{1+C^2}{2}\frac{\lambda}{1-\lambda}.$$

Again, we observe a $(1+r)$-fold reduction in queueing overhead.

The comparison with $SQ(2)$ in explicit forms is not included here, as no explicit expression is available for mean service time of $SQ(2)$ with general service time distributions. For the `JIQ-SQ(2)` algorithm, its queueing overhead is also difficult to characterize explicitly. However, `JIQ-SQ(2)` is expected to show more significant reduction than `JIQ-Random`. The performance of `JIQ-SQ(2)` is shown with ρ obtained numerically in Fig. 3.3.



(a) (b)

Figure 3.3: (a) Comparison of the computed mean response time for `Random`, `SQ(2)` and `JIQ-Random` with $r = 10$, 20 and 40. (b) Comparison of the mean response time for `Random`, `SQ(2)` and `JIQ-SQ(2)` with $r = 10$, 20 and 40. Both figures are for PS service discipline and general service time distributions.

Figure 3.3 illustrates the comparison of `Random`, `SQ(2)` and `JIQ` algorithms with different r. At low load, `JIQ` and `SQ(2)` algorithms result in similar mean response time. This is because `SQ(2)` is sufficient to assign most arrivals to idle servers when the load is low. However, `JIQ` outperforms `SQ(2)` significantly at medium to high load. When the load is extremely high, the advantage of `JIQ` diminishes since most I-queues are empty.

3.2 Evaluation via Simulation

We evaluated the class of JIQ algorithms against SQ(d) for a variety of service time distributions in [37]. The performance of JIQ algorithm is summarized as follows:

1. JIQ-Random outperforms SQ(2).

It showed that JIQ-Random algorithm outperforms SQ(2) significantly at medium to high load in terms of average response time. For instance, the JIQ-Random algorithm with $r = \frac{n}{m} = 40$ achieves a 7−fold reduction of queueing overhead from the SQ(2) algorithm, for the PS discipline at load 0.9.

2. JIQ-SQ(2) achieves near-optimality.

Simulation results show that the JIQ-SQ(2) algorithm achieves close to minimum response time.

3. The JIQ algorithms are near-insensitive.

Under PS service discipline, the JIQ algorithms are shown to be near-insensitive to the service distributions.

Remak: In addition to the shorter response time, the proposed JIQ algorithm also shows its advantage over the SQ(d) algorithm in terms of complexity, as it incurs no communication overhead at request arrivals.

3.3 Extension of JIQ

So far, we have shown the advantage of the JIQ algorithms for large-scale homogeneous systems with distributed dispatchers via theoretical analysis and simulation. The JIQ algorithm holds great promise for being implemented in real world systems. Every opportunity is paired with a challenge. The gap between the complexity of real systems and the simplicity of the model we used is significant. This poses challenges in making the JIQ algorithms applicable in the real word with further extension.

There are many interesting problems to investigate.

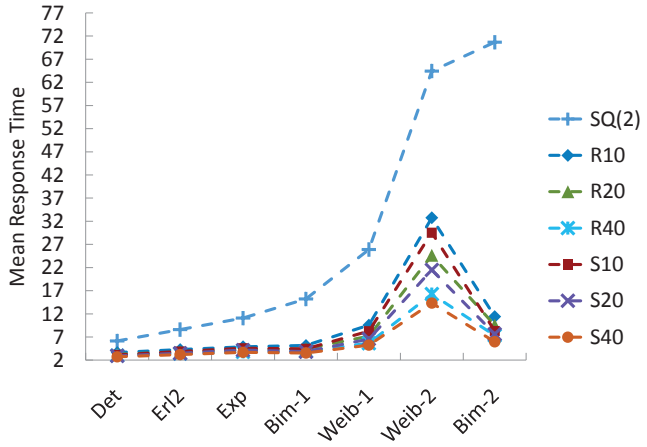


Figure 3.4: Mean response time comparison for `SQ(2)`, `JIQ-Random` and `JIQ-SQ(2)` with reporting threshold equal to two, with 7 different service time distributions at load of 0.99. The smallest possible mean response time is 2 with a mean service time of 2.

3.3.1 Varying report threshold

As pointed out in the previous section, the `JIQ` algorithms have no advantage over `SQ(d)` at very high load. This is because the number of idle servers in the system becomes small at high load and I-queue is empty with high probability. In [37], we proposed changing report threshold dynamically. At very high load, a server informs the I-queues when it is lightly loaded. For instance, a server can report to one I-queue when its queue length is decreasing from two to one. And it reports again when it becomes idle. This will insert one copy of all servers with one job and two copies of all idle servers in the I-queues, and further increase the arrival rate to idle servers with zero or one job.

We evaluate the extension of the `JIQ` algorithms with reporting threshold equal to two at a high load of 0.99. This is the region where the performance of the original `JIQ` algorithms is similar to that of `SQ(2)`. However, with reporting threshold equal to two, the `JIQ` algorithms significantly outperforms `SQ(2)`, shown in Fig. 3.4. For instance, with exponential distribution, for which service disciplines do not affect response times, `SQ(2)` outperform `JIQ-Random` with threshold equal to one in Fig. 3.3, but is outperformed by `JIQ-Random` with $r = 10$ and threshold equal to two, with 88% reduction in queueing overhead.

An interesting observation is that the mean queue sizes are no longer mono-

tonically increasing with variance of service times. In particular, the two bimodal distributions have smaller mean queue sizes than distributions with smaller variance. For the bimodal distributions with variance 99, `JIQ-Random` with $r = 10$ reduces the mean queue size from that of `SQ(2)` by 89%, and `JIQ-SQ(2)` with $r = 40$ reduces the mean queue size from that of `SQ(2)` by 97.1%. On the other hand, for the Weibull distribution with variance 76, `JIQ-SQ(2)` with $r = 40$ reduces the mean queue size from that of `SQ(2)` by only 83%. Apparently higher moments of the distribution start to have an effect when the reporting threshold is more than 1.

To determine the reporting threshold, the rule of thumb is half of the resulting mean queue size. Decreasing the reporting threshold will increase the rate of random arrivals, which results in a larger queue size. On the other hand, increasing the reporting threshold will attract too many arrivals at once to an idle server and result in unbalanced load. The mean queue size with a threshold other than one remains difficult to analyze.

It is natural to ask: can we control the report parameter in other ways? As our goal is to minimize response time, average response time of arrivals at a server might be a better criterion for reporting. A better understanding of the algorithm with varying reporting threshold is necessary before we move on to further design and analysis.

3.3.2 Heterogeneous system

For a large data center that consists of tens of thousands of machines, the heterogeneity of server nodes comes from many aspects: different network interface speeds, different numbers of processors and memory sizes. There has been a large amount of work on the load balancing for a nonhomogeneous multiple servers system with a central job dispatcher [40, 41, 42]. Existing algorithms with a centralized design incur high communication overhead for distributed dispatchers. As far as we know, there has been no study on load balancing for a heterogeneous servers system with distributed dispatchers.

It is interesting to evaluate the performance of the basic version of the `JIQ` algorithm in a heterogeneous system and further extend it to achieve better performance.

Consider a simple heterogeneous system which comprises two kinds of par-

allel servers: fast servers with higher service rate and slow servers with lower service rate. In the basic version of JIQ, arrivals at dispatchers with empty I-queue are directed to a server selected uniformly randomly. Since servers present at I-queue are idle, it is still reasonable to assign arrivals to these servers if any exists at a dispatcher. An idea to modify the original JIQ algorithm for heterogeneous systems is to change the assignment strategy of arrivals that see empty I-queues.

In [40], an algorithm of probabilistic splitting Poisson arrivals is proposed to minimize average job response time for a heterogeneous multiserver system with a central job scheduler. If dispatchers have the knowledge of servers service rate, weighted assignment of arrivals seeing empty I-queues might be a promising strategy.

What if servers have no idea of a server's property? Intuitively, with same load, fast servers becomes idle more frequently than slow servers. In other words, the frequency of a server's presence in the I-queue indicates its service rate. By making use of the information at I-queue, the dispatchers might wisely assign more requests to fast servers without extra communication to acquire a server's service rate.

3.3.3 Accuracy of JIQ-SQ(2)

JIQ-SQ(2) is proved to achieve near-optimality, which makes it promising for implementation. With SQ(2) for the secondary load balancing, when a server becomes idle, it needs to inquire with two I-queues and wait for the response to make a report decision. Although the delay for the communication between dispatchers and servers is small, the retrieval of inquiry response at servers might take quite some time. The delay makes it possible that, at the moment of an idle server's making a report decision, the I-queue sizes sampled by the servers are out of date, which may result in wrong report decision.

An approach to this problem is to apply work stealing/sharing among dispatchers. When a server becomes idle, it joins a randomly selected I-queue. Then try to balance idle servers at I-queues by work stealing/sharing. When load for the system is low, it is rare that I-queues are empty. Work stealing is preferred. On the contrary, empty I-queues are common at high

load, so work sharing might be more efficient.

Further effort is required on the design and analysis of the work stealing/sharing strategy for our model. Here is a rough idea: when an I-queue size exceeds certain threshold, it tries to push some idle servers to other I-queues.

CHAPTER 4

BACK-END TASK ASSIGNMENT WITH DATA LOCALITY CONSTRAINT

In this chapter, we focus on the back-end task assignment with data locality constraint. Data locality is an important problem as it significantly affects system throughput and job completion times. There has been a large amount of recent work on task assignment and data placement algorithms [43, 19, 44]. However, to the best of our knowledge, there has been no analytical understanding of the effect of various algorithms on throughput and delay of the system. Here a discrete time model is set up for algorithm analysis and a new algorithm is proposed.

4.1 Discrete-Time Model

In this research, we consider a discrete-time assignment model with uniformly random data placement, and different service rates for computations with and without local data.

Consider a discrete-time model for a system with m parallel servers. At the beginning of each time slot, a constant number of tasks arrive at the system. Each task processes one file chunk, which is replicated d times and placed on d randomly selected servers. We do not consider the constraint of network structure on data placement. With the constant arrivals, we are interested in studying system throughput and task completion times at a fixed load. We defer to future work the study of job completion times when each job spawns a large number of tasks, and a job completes only when all its tasks complete.

The placement of data can be modelled by bipartite graphs G with n task nodes and m server nodes, as illustrated in Fig. 4.1. An edge between task node i and server node j indicates the presence of data for task i on server j. And we define the degree of a server node as the number of unassigned tasks

that have data on it. In particular, we consider the default data replication scheme of 3 replicas for each data chunk, i.e., $d = 3$. The scheme can be easily extended to a variable number of replicas for different data chunks, as proposed in [43, 44].

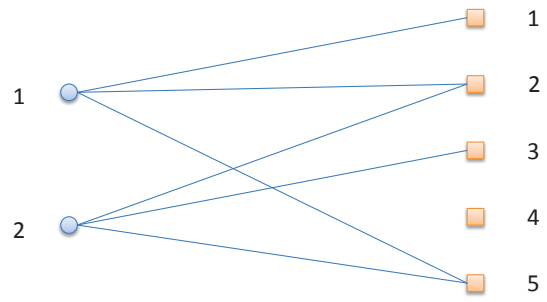


Figure 4.1: Example of a random graph with $n = 2$, $m = 5$, $d = 3$.

Assume that each server can process only one task at any time. Task assignment is performed at the beginning of an interval and each server can process only one task at any time. All tasks are assigned unless there are no more idle servers, and the remaining tasks are kept in a queue. When a task is assigned to a server with local data (i.e., an edge exists in G), it leaves the system with probability p at the end of an interval; when it is assigned to a server with remote data (i.e., no edge exists in G), it leaves the system with probability $q < p$. Hence the service times follow a geometric distribution with mean $\frac{1}{p}$ with local data and a geometric distribution with a larger mean $\frac{1}{q}$ with remote data. We refer to servers serving tasks with local data as *p-server* and servers serving tasks without local data as *q-server*.

4.2 Algorithm Description

The objective of task assignment is to assign as many tasks as possible to a server with local data, which is actually a matching problem. We consider two assignment algorithms. The first algorithm models the current scheduling algorithm in production clusters. And we proposed the second algorithm in [45].

1. Random server algorithm: Whenever there are outstanding tasks in the system, an idle server is chosen randomly. If there exist multiple tasks

whose data is replicated on this server, one task is uniformly selected from the set. If no outstanding task has local data on the server, this server is assigned for a randomly selected task. This models the FIFO scheduler currently used in Hadoop clusters [13, 44], which assigns tasks following exactly the same rule. The random selection of servers models the continuous-time effect where servers become idle and request tasks in sequence, without collaboration of other servers.

2. Degree-guided algorithm: An idle server with the least non-zero degree is sampled if outstanding tasks are present. And this server is assigned for a task that is randomly selected from all of its connected tasks. When all idle servers are of degree 0, a server is selected randomly for a random task. The least non-zero degree of the selected server ensures that the task is assigned to a p-server. In addition, it maintains the connections of unassigned tasks to the remaining idle servers to the utmost extend, which increases the probability of assigning these tasks to p-servers.

In this research, we consider a simple version of the degree-guided algorithm, called *Peeling algorithm*. With the Peeling algorithm, an idle server that has local data for only one outstanding task is assigned to this task. The procedure continues until no server of degree 1 exists, which is referred to as the *peeling stage*. Then the assignment procedure continues with the Random Server algorithm, which is called the *random stage*. Note that the Peeling algorithm is equivalent to the degree-guided algorithm until the peeling stage stops.

4.3 Mean-Field Model for a Single Time Slot

In this section, we focus on the analysis of the Random Server algorithm and the Peeling algorithm over one time slot. We need the following definitions to derive the mean-field models for the two algorithms.

Definition 1 Degree Distributions from A Node Perspective. *Given a graph G with n task nodes and m server nodes, let L_i denote the number of task nodes of degree i, $i = 0, 1, \cdots, l_{max}$, and R_j the number of server nodes of degree j, $j = 0, 1, \cdots, r_{max}$, where l_{max} and r_{max} are the largest degree of task nodes and server nodes respectively. So $\sum_i L_i = n$ and $\sum_i R_i = m$. The*

degree distributions from a node perspective are defined by the pair (L, R)*, where* $L = \{L_0, L_1, \cdots, L_{l_{max}}\}$ *and* $R = \{R_0, R_1, \cdots, R_{r_{max}}\}$*.*

Definition 2 The Standard Ensemble $\mathbf{G(L, R)}$. *Given the degree distributions* (L, R)*, we define an ensemble of bipartite graphs* $G(L, R)$ *in the following way. Each graph in* $G(L, R)$ *has the degree distributions* (L, R)*. As a node of degree* i *has* i *sockets from which the* i *edges emanate, there are a total of* $\sum_i iL_i = \sum_j jR_j$ *sockets on each side. Label the sockets on each side with numbers* $\{1, \cdots, \sum_i iL_i\}$*. Let* σ *be a permutation on* $\{1, \cdots, \sum_i iL_i\}$*, and associate* σ *to a bipartite graph by connecting the* j*-th socket on the task nodes to the* $\sigma(j)$*-th socket on the server nodes. We define a probability distribution over the set of graphs by placing the uniform probability distribution on* σ*. The definition is the same as the standard ensemble for LDPC codes [46], pg 78.*

We model the assignment procedure as an evolution of the random graph ensemble. Let $s \in \mathcal{N}$ denote the assignment step, which starts at zero and increases by one for every task assigned. At each step, all edges connected to the selected server and those connected to the assigned task are removed from the graph. This procedure results in a sequence of *residual* graphs, denoted by $G(L(s), R(s))$, where $(L(s), R(s))$ are the degree distributions of unsigned tasks and remaining idle servers at step s. No edge is left at the end of the assignment as either all tasks are assigned or no idle server remains.

Consider the graph evolution at each time step for both the Random Server algorithm and Peeling algorithm:

Random Server Algorithm: A server node is randomly selected at each time step. If the server node has degree 0, a task is sampled from the unassigned pool uniformly randomly and assigned to the server. All edges connected to the assigned task are removed from the graph. If the server node has degree $i > 0$, a task is sampled from the i connected tasks uniformly randomly and assigned to the server. All i edges connected to the server, and all edges connected to the assigned tasks are removed from the graph.

Peeling Algorithm: A server node with degree 1 is selected at each time step. The only task connected to the server node is assigned to the server. All edges connected to this task are removed from the graph. The algorithm pauses if no server node with degree 1 exists.

Consider a system with n unassigned tasks and k idle servers before the assignment ($k \leq m$). Then $\frac{L_i(0)}{n}$ follows a binomial distribution $B(d, \frac{k}{m})$ and $\frac{R_i(0)}{k}$ a binomial distribution $B(dn, \frac{1}{m})$ truncated at r_{max}. Hence $l_{max} = d$ and we can set r_{max} a fixed constant as there is a limited amount of storage space on each server. Let $M_p(s)$ and $M_q(s)$ denote the increased numbers of p-servers and q-servers at the end of step s respectively. Define the scaled time $\tau = \frac{s}{w}$, where $w = nd$ denotes the total number of edges in the initial graph. Let $\gamma_i(\tau) = \frac{R_i(w\tau)}{w}$, $l_i(\tau) = \frac{L_i(w\tau)}{w}$, $m_p(\tau) = \frac{M_p(w\tau)}{w}$, $m_q(\tau) = \frac{M_q(w\tau)}{w}$, which together determine the *assignment path* for a single slot. We obtain the following theorem. Full proof can be found in Appendix A.

Theorem 3 Evolution of Residual Graph for the Assignment Algorithms. *The expected assignment paths of the two algorithms are described by the two sets of differential equations respectively:* Random Server algorithm:

$$\begin{aligned}
\frac{dl_i(\tau)}{d\tau} &= -\frac{l_i\gamma_0}{v(\tau)c(\tau)} + \sum_{j\geq 1}\frac{\gamma_j}{c(\tau)}\frac{(j-1)(i+1)l_{i+1} - jil_i}{e(\tau)}, \\
&\quad for\ \ 0 \leq i \leq l_{max} - 1, && (4.1)\\
\frac{d\gamma_i(\tau)}{d\tau} &= -\frac{\gamma_i}{c(\tau)} + [(i+1)\gamma_{i+1} - i\gamma_i][\frac{\gamma_0}{v(\tau)c(\tau)} + (1 - \frac{\gamma_0}{c(\tau)})\frac{\sum_{s\geq 1} s(s-1)l_s}{e^2(\tau)}], \\
&\quad for\ \ 0 \leq i \leq r_{max} - 1, && (4.2)\\
\frac{dm_p(\tau)}{d\tau} &= 1 - \frac{\gamma_0}{c(\tau)}, && (4.3)\\
\frac{dm_q(\tau)}{d\tau} &= \frac{\gamma_0}{c(\tau)}, && (4.4)
\end{aligned}$$

where $v(\tau) = \sum_j l_j(\tau)$, $c(\tau) = \sum_j \gamma_j(\tau)$, *and* $e(\tau) = \sum_j jl_j(\tau) = \sum_j j\gamma_j(\tau)$.

Peeling algorithm:

$$\begin{aligned}
\frac{dl_i(\tau)}{d\tau} &= -\frac{il_i}{\sum_j jl_j(\tau)}, \quad for\ \ 2 \leq i \leq l_{max} - 1, && (4.5)\\
\frac{d\gamma_1(\tau)}{d\tau} &= (2\gamma_2 - \gamma_i)\sum_k \frac{(k-1)kl_k}{\sum_j jl_j(\tau)} - 1, && (4.6)\\
\frac{d\gamma_i(\tau)}{d\tau} &= ((i+1)\gamma_{i+1} - i\gamma_i)\sum_k \frac{(k-1)kl_k}{\sum_j jl_j(\tau)}, \\
&\quad for\ \ 2 \leq i \leq r_{max} - 1, && (4.7)\\
m_p(\tau) &= \tau \quad and \quad m_q(\tau) = 0, && (4.8)
\end{aligned}$$

where $v(\tau), c(\tau)$ and $e(\tau)$ are defined as above.

With probability at least $1 - O(n^{\frac{1}{6}} e^{-\frac{\sqrt{dn}}{c^3}})$, the assignment path of a specific instance has maximum L_1-distance from the expected assignment path at most $O(n^{-\frac{1}{6}})$ from the start of the process until either the total number of nodes in the residual graph has reached size ηn, where η is an arbitrary strictly positive constant, or the algorithm gets stuck.

By solving these differential equations, we can obtain the expected increased fractions of p-servers and q-servers after the assignment, which provide an efficient way to evaluate the performance of these two algorithms.

4.4 Performance for a Single Time Slot

Let (f_i, f_p, f_q) denote the fraction of idle servers, p-servers and q-servers at the beginning of a time slot, hence $f_i, f_p, f_q \in [0, 1]$ and $f_i + f_p + f_q = 1$. We first show the property of function $\sigma_p(r, f_i) = \frac{M_p(n)}{m}$, which denotes the increased fraction of p-servers at the end of assignment process for a system with $\frac{n}{m} = r$ and f_i percent of idle servers before the assignment. We then evaluate the performance of the two algorithms against the maximum matching, which is optimal.

4.4.1 Properties of the function $\sigma_p(r, f_i)$

First, we show that for the Random Server algorithm, the function $\sigma_p(r, f_i)$ does not depend on the fraction of idle servers available at the beginning of the slot, so long as all tasks are assigned eventually.

Theorem 4 Independence of f_i. *Consider the Random Server algorithm for a system with m servers and $n = mr$ unassigned tasks. Let the fraction of servers be (f_i, f_p, f_q) before the assignment. If $r < f_i$, the increased fraction of p-servers, $\sigma_p(r, f_i)$, is independent of f_i.*

Proof. For the Random Server algorithm, the evolution of the assignment process only depends on the initial graph connecting the tasks and the idle servers, and the sequence of idle servers sampled. Denote the initial graph

connecting n tasks and k idle servers by $G_{n,k}$, where the degree distributions follow the binomial distribution specified in Section 4.3, and denote the sequence of idle servers sampled $I_{n,k} = \{i_h, h = 1, \cdots, n : 1 \leq i_h \leq k\}$.

Consider the case when all servers are idle, that is, $k = m$. Since the sequence of sampled idle servers $I_{n,m}$ is independent of the residual graph at step t, the evolution remains the same if we determine $I_{n,m}$ before the assignment process starts. Restrict the graph $G_{n,m}$ to the n server nodes in $I_{n,m}$ and denote it by G'_n, where each server node retains its original index. Observe that with $I_{n,m}$ fixed, the number of p-servers assigned is determined by the graph G'_n.

Next consider the case when a fraction f_i of servers are idle. Using the same argument above, we can determine the sequence of sampled idle servers I_{n,mf_i} before the assignment process, and restrict the graph G_{n,mf_i} to the n sampled server nodes. Denote the graph by $\hat{G}_n$. With I_{n,mf_i} fixed, the number of p-servers assigned is determined by the graph $\hat{G}_n$.

For both G'_n and $\hat{G}_n$, the degree distribution of the task nodes follows the binomial distribution $B(d, f_i)$ and that of the server nodes follows the binomial distribution $B(nd, \frac{1}{m})$. Hence the ensembles G'_n and $\hat{G}_n$ have the same distribution. Conditioned on the n servers sampled, the sequences $I_{n,m}$ and I_{n,mf_i} are samples from the uniform distribution on permutations on the n servers, hence they have the same distribution. This yields exactly the same number of p-servers assigned in a finite system. Hence $\sigma_p(r, f_i)$ is independent of f_i. ■

Figure 4.2 shows the plots of $\sigma_p(r)$ for the Random Server algorithm and the plots of $\sigma_p(r, f_i)$ for the Peeling algorithm with different values of f_i. Note that $\sigma_p(r, f_i) = \frac{M_p(n)}{m}$ can be obtained by solving the differential equations in Theorem 3 numerically. At the same load, the increased fraction of p-servers by the Peeling algorithm is larger than the Random Server algorithm.

Note that for Peeling algorithm with $f_i = 1$, there exists an obvious point for r, above which $\sigma_p(r, f_i)$ decreases before increases again. Below the threshold, $\gamma_1(\tau) > 0$ for $\tau \in [0, min\{\frac{1}{d}, \frac{f_i}{dr}\}]$. That is, servers of degree-1 are available throughout the assignment procedure so the peeling stage doesn't stop. Hence $\sigma_p(r, f_i) = min\{r, f_i\}$, which equals r if $r < f_i$. With $f_i < 1$, however, some tasks are only connected to occupied servers, which will result in the emergence of random stage and hence decrease σ_p.

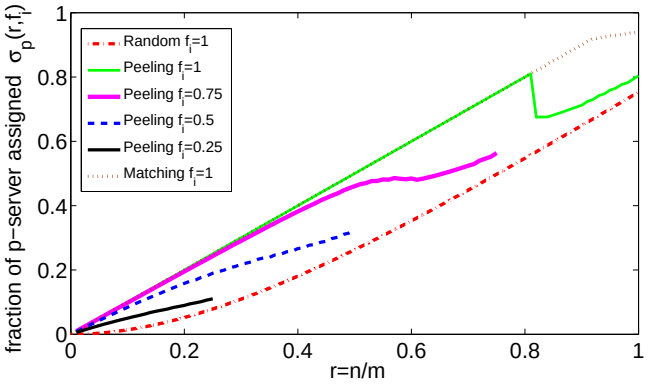


Figure 4.2: Plots of $\sigma_p(r, f_i)$ with $d = 3$, for the Random Server algorithm and the Peeling algorithm with different fraction of idle servers, f_i, at the beginning of the assignment process.

We also observe that $\sigma_p(r)$ increases monotonically for the Random Server algorithm. We have the theorem below.

Theorem 5 Monotonicity. *Consider the Random Server algorithm for a system with m servers and $n = mr$ tasks arriving. Let $f_i = 1$ and $r < f_i$. Then the increased fraction of p-servers, $\sigma_p(r)$, strictly increases with r.*

Proof. We show that $\sigma_p(\frac{n}{m}) < \sigma_p(\frac{n+1}{m})$ for $n, m \in N$ using coupling. In particular, let $G_{n,m}$ be the random graph whose degree distributions follow the binomial distributions specified in Section 4.3. We couple the random graph $G_{n,m}$ with the graph $G_{n+1,m}$ such that they differ only in the extra task node and its d edges. We reuse the notation σ_n to denote the fraction of p-servers assigned with the instance of $G_{n,m}$ on a particular sample path. The theorem is proven if we show that for every pair of graphs G_n and G_{n+1} where the latter has one extra task node with $s > 0$ edges, but are otherwise the same, $\sigma_n < \sigma_{n+1}$. We use the induction method.

Initial condition. We have $\sigma_0 = 0$ by definition, and $\sigma_1 = \frac{s}{m} > 0$, which is the probability that one of the d server nodes connected to the single task node is chosen. Hence $\sigma_0 < \sigma_1$ for all G_n and G_{n+1}.

Induction step. Assume that for all $k < n$, $k, n \in N$, $\sigma_k < \sigma_{k+1}$ for all G_k and G_{k+1}. Consider an arbitrary pair G_n and G_{n+1}, we want to show that $\sigma_n < \sigma_{n+1}$. We consider the regions $\frac{n+1}{m} \leq 1$ and $\frac{n+1}{m} > 1$ separately.

First we consider the region with $\frac{n+1}{m} \leq 1$, so that all tasks are assigned. With the standard coupling, the same sequence of server nodes are chosen for G_n and G_{n+1} except for the last step where an extra server is needed for the latter. Also, the random selection of task nodes are coupled: for a server node with the same non-zero degree in G_n and G_{n+1}, the same task will be assigned; For a server node with non-zero degree in G_n and G_{n+1}, but connected to the extra task node in G_{n+1}, and for a degree-zero server node, either the same tasks are assigned in G_n and G_{n+1}, or the extra task is assigned in G_{n+1}. Let σ_n^i denote the fraction of p-servers assigned up to step i for G_n. We consider two cases:

Case 1: At each assignment step, the server node in G_{n+1} selects the same task node as that in G_n. Hence after n steps, $\sigma_n^n = \sigma_{n+1}^n$. In the graph G_{n+1}, the extra task is assigned to a server with local data with probability $\frac{s}{m}$. Hence $\sigma_{n+1} = \sigma_{n+1}^{n+1} > \sigma_n^n = \sigma_n$.

Case 2: Let $i < n+1$ be the first assignment step where the selected server node in G_{n+1} selects the extra task node, denoted by T_{n+1}, and the corresponding server node in G_n selects some other task denoted by T_i. We have $\sigma_n^i = \sigma_{n+1}^i$ by the definition of i. Consider the remaining graphs G_{n-i} and G_{n-i+1}, which are the same except for an extra task node T_i in G_{n-i+1}. With a positive probability the task node T_i is connected to s idle server, where $s > 0$. The induction assumption states that $\sigma_{n-i} < \sigma_{n-i+1}$ for all pairs G_{n-i} and G_{n-i+1}. Hence $\sigma_n < \sigma_{n+1}$.

We have completed the proof for the region $\frac{n+1}{m} \leq 1$.

For the region $\frac{n+1}{m} > 1$, we consider the same two cases as above. In Case 1, $\sigma_n = \sigma_{n+1}$ as the extra task node is not assigned. Case 2 is handled in exactly the same way, and we obtain $\sigma_n < \sigma_{n+1}$ like before. This completes the proof of Theorem 5. ■

4.4.2 Comparison with maximum matching

The objective of task assignment is to assign as many tasks as possible to a server with local data, which is a matching problem. The following theorem shows the performance of the maximum matching algorithm, which is optimal.

Theorem 6 *[47] Consider a system where $n, m \to \infty$ with $r = \frac{n}{m}$ fixed and*

each task has its data on d randomly selected servers. Let y be the unique solution to the equation:

$$d = \frac{y(1 - e^{-y})}{1 - e^{-y} - ye^{-y}} \tag{4.9}$$

and $r^* = \frac{y}{d(1-e^{-y})^{d-1}}$. *If* $r \leq r^*$, *all tasks are assigned to p-servers; if* $r \geq r^*$, *proportion of p-servers assigned is:*

$$\sigma_p \to r + 1 - e^{-z} - ze^{-z} - \frac{z}{d}(1 - e^{-z}), \tag{4.10}$$

where $z = drx$ *and* x *is the largest solution to*

$$x = (1 - e^{-drx})^{d-1}. \tag{4.11}$$

For the case of $d = 3$, we obtain $r^* = 0.918$. The performance of maximum matching is showed in Fig. 4.2. We can see that with all servers idle initially, the Peeling algorithm achieves the optimal performance as maximum matching, if r is below the threshold for the emergence of the random stage. And the improvement of the Peeling algorithm over the Random Server algorithm is significant. Above the threshold, the Peeling algorithm deteriorates from the optimal matching, but still performs better than the Random Server algorithm. We have the following lemma.

Lemma 3 *The Peeling algorithm achieves the performance of optimal matching when the load for the system is below the threshold for the emergence of the random stage.*

4.5 Fixed Point Characterization

In this section, we characterize the fixed points of the system evolution when the load is below the threshold that no task remains in the queue after the assignment.

Consider a system with $r = \frac{n}{m}$ fixed and $n, m \to \infty$. Let (π_i, π_p, π_q) denote the equilibrium values of (f_i, f_p, f_q) before the assignment without tasks queueing, and $(\hat{\pi}_i, \hat{\pi}_p, \hat{\pi}_q)$ the equilibrium values after the assignment.

Theorem 7 Fixed point characterization. *For both the Random Server algorithm and the Peeling algorithm, define*

$$g(\pi_i) = 1 - \frac{\sigma_p(r,\pi_i)}{p} - \frac{r - \sigma_p(r,\pi_i)}{q},$$

where $g(\pi_i)$ is different for these two algorithms. If no queueing takes place, π_i satisfies

$$\pi_i = g(\pi_i) + r \quad \textit{and} \quad g(\pi_i) > 0. \tag{4.12}$$

And the fixed point is

$$(\pi_i, \pi_p, \pi_q) = \left(\pi_i, \frac{(1-p)\sigma_p(r,\pi_i)}{p}, \frac{(1-q)(r-\sigma_p(r,\pi_i))}{q}\right),$$

$$(\hat{\pi}_i, \hat{\pi}_p, \hat{\pi}_q) = \left(g(\pi_i), \frac{\sigma_p(r,\pi_i)}{p}, \frac{r-\sigma_p(r,\pi_i)}{q}\right).$$

Proof. With $n = mr$ and no tasks in the queue, at the fixed point, all n tasks are assigned, which yields the following:

$$\begin{aligned}
\hat{\pi}_p &= \pi_p + \sigma_p(r,\pi_i), \\
\hat{\pi}_q &= \pi_q + r - \sigma_p(r,\pi_i), \\
\hat{\pi}_i &= \pi_i - r.
\end{aligned}$$

With the geometric distribution for the service time, a task leaves a p-server with probability p and leaves a q-server with probability q, we have

$$\begin{aligned}
\pi_i &= \hat{\pi}_i + \hat{\pi}_p p + \hat{\pi}_q q, \\
\pi_p &= \hat{\pi}_p (1-p), \\
\pi_q &= \hat{\pi}_q (1-q).
\end{aligned}$$

Solving the equations, we obtain

$$\pi_i = g(\pi_i) + r.$$

To ensure all tasks assigned, we need $\pi_i > r$, which yields Eq. (4.12). Substituting π_i in the above equations yields the fixed point. ∎

Remark. For the Random Server algorithm with $\pi_i > r$, Theorem 4 indicates that $\sigma_p(r,\pi_i)$ only depends on r. So the computation of its fixed point

can be simplified.

4.5.1 Queueing threshold

From Theorem 7, we can define the threshold for queueing.

Definition 3 Threshold for Queueing. *We define the threshold of load below which no queueing takes place as*

$$\rho^*(p,q) = \frac{\sup\{r \in [0,1] : \pi_i = g(\pi_i) + r \quad and \quad g(\pi_i) > 0\}}{p}.$$

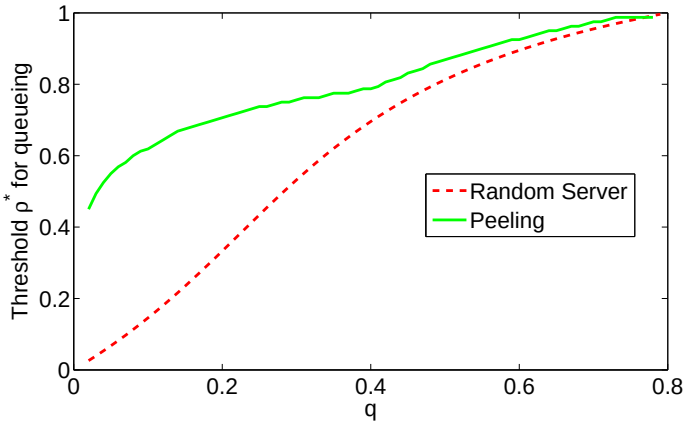


Figure 4.3: Threshold for queuing for the Random Server algorithm and the Peeling algorithm with $p = 0.8$.

By solving the mean fields equations in Theorem 3, we obtain a table of $\sigma_p(r, f_i)$. With the values of $\sigma_p(r, f_i)$, we can obtain the queueing threshold ρ^*. For instance, with $p = 0.8$ and $q = 0.4$, $\rho^* = 0.695$ for the Random Server algorithm and $\rho^* = 0.765$ for the Peeling algorithm. Figure 4.3 plots the thresholds for these two algorithms against q with $p = 0.8$. It shows that with the same p and q, the threshold under the Random Server algorithm is smaller than that under the Peeling algorithm. In addition, the thresholds for both algorithms increase to 1 as q increases towards p, since the system behaves as a homogeneous system when q almost equals p.

4.5.2 Effective mean service rate

We define the effective mean service rate μ_e as

$$\mu_e = \frac{n}{m\pi_p + m\pi_q} = \frac{\lambda}{\pi_p + \pi_q}, \tag{4.13}$$

which measures the efficiency of the servers. When the load is below the queueing threshold, μ_e can be obtained from Theorem 7.

We do not have explicit characterization of the fixed points at high load. Instead, we iterate the mean-field equations over multiple time slots to obtain the fixed point. In addition to the degree distribution of the graph connecting unassigned tasks and idle servers, we also need those of the graphs connecting unassigned tasks and p-servers, and unassigned tasks and q-servers. The evolution equations are derived in a similar way as Theorem 3. We defer the evolution equations and proof to Appendix B. At the beginning of a time slot, the degree distributions of the three graphs are obtained by convoluting those due to tasks in the queue and those due to new arrivals. The assignment algorithm is run on the graph with idle servers, and all degree distributions are updated accordingly. The characterization of the fixed point is difficult. This is because the degree distributions of the unassigned tasks at the end of a time slot are not Poisson and hard to track. The mean-field equations are iterated over multiple periods until a fixed point is reached.

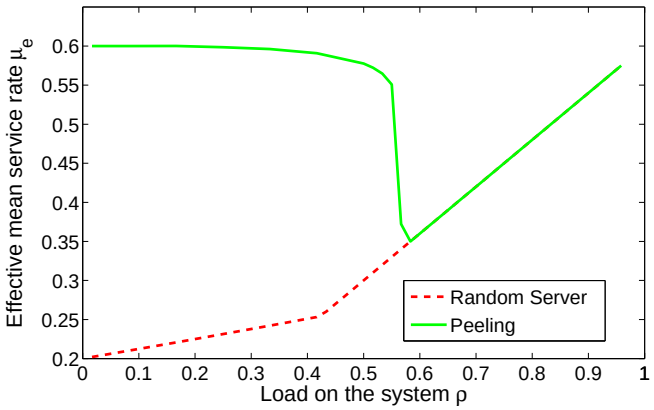


Figure 4.4: Effective mean service rate for the Random Server algorithm and the Peeling algorithm (maximum possible service rate is 0.6) with $p = 0.6$ and $q = 0.2$. Note that $\rho^* = 0.417$ for the Random Server algorithm and $\rho^* = 0.55$ for the Peeling algorithm.

Figure 4.4 shows the effective mean service rate at the fixed points for both algorithms with $p = 0.6$ and $q = 0.2$. As soon as queueing takes place, the mean service rate for the Peeling algorithm decreases drastically and converges to that of the Random Server algorithm. This is due to the lack of degree-one server nodes and the peeling stage stops when there are still a large number of unassigned tasks.

4.6 Performance in Continuous Time Model

We evaluate the Peeling algorithm against the Random Server algorithm via simulation in continuous time. Consider a system of m parallel servers. Tasks arrive at the system as a Poisson process with rate-$m\lambda$. The service time of a task assigned to a server with (without) local data is assumed to i.i.d. with distribution $B_p(\cdot)$ ($B_q(\cdot)$) with mean $\frac{1}{p}$ ($\frac{1}{q}$). Tasks are assigned in the following ways:

Random Server algorithm: When an arriving task sees some idle servers, an idle server is randomly selected for this task; otherwise this task joins the queue. When a server becomes idle, if its degree is zero, a task is sampled from the unassigned pool uniformly randomly; otherwise a task is selected randomly from the tasks that have data replicated on this server. If no unassigned task exists in the system, the server just stays idle.

Peeling algorithm: When a task arrives, if no idle servers are available, it joins the queue; otherwise a server with the least non-zero degree is selected to process this task. If all idle servers are of degree 0, this tasks is assigned to a randomly selected idle server. When a server becomes idle, it follows the same rule as the Random Server algorithm.

Consider exponential service time distribution. Figure 4.5 shows the effective mean service rate. Similar to the results in Fig. 4.4, which is obtained from the computation using mean-field equations in the discrete-time model, the Peeling algorithm outperforms the Random Server algorithm at low to moderate load, while it converges to the Random Server algorithm with load increasing. Note that these two algorithms show similar performance trends in different models. Hence performance analysis under the discrete-time model provides insight for the performance of these algorithms in real scenario, and also offers guidelines for the design of efficient algorithms.

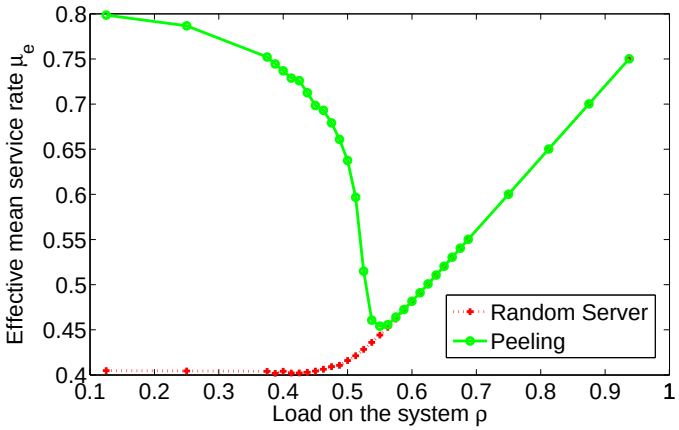


Figure 4.5: Effective mean service rate for the Random Server algorithm and the Peeling algorithm (maximum possible service rate is 0.8) with $p = 0.8$, $q = 0.4$, $m = 200$ and $d = 3$.

CHAPTER 5

CONCLUSION

This thesis investigated two challenges of the Cloud service data centers to achieve high performance: the front-end load balancing and the back-end task assignment. Today the workload of web-service requests is highly dynamic. To achieve service provision and cost efficiency, Cloud service data center should be designed with scalability. This motivated the distributed design of dispatchers for the front-end system. In this research, a class of novel algorithms, `Join-Idle-Queue` (JIQ), is proposed for large scale load balancing with distributed dispatchers. Both theoretical analysis and simulations have been performed to show the advantage of `JIQ` algorithms:

- `JIQ` algorithms result in shorter response time than existing algorithms including `SQ` and `Random` algorithms. The actual response time is further reduced with `JIQ`, as it eliminates extra communication at requests arrival.
- With PS service discipline, the mean response time produced by `JIQ` is shown to be insensitive to service time distributions

For the back-end system, fast information retrieval from large amount of data is the main issue. Cluster computing, such as Map-Reducee and Hadoop, has been widely used to achieve parallel processing. This research focused on the Map-Reduce task assignment with data locality, which significantly affects system throughput and job completion times. A degree-guided algorithm is proposed in this study. And it has been shown to produce higher effective service rate than current scheduling algorithms in the region of light to medium loads. In particular, an analytical model has been established for the Map-reduce task assignment problem with data locality. And the accuracy of this analytical model is confirmed with simulation.

A common feature shared by the `JIQ` load balancing algorithm and the degree-guided task assignment algorithm is that performance at high load is

not satisfactory. In Chapter 3, an extension of JIQ for high load has been discussed. As pointed out, more work is still needed. One step for future work is to explore the extension of these two algorithms for high load. And it would also be interesting to investigate the potential of JIQ for heterogeneous systems. In this research, the evaluation of algorithm performance is based on both analysis and simulation. As both the front-end and back-end systems are in large scale, hundreds of servers are needed to set up a testbed. These facilities are not available at the moment. The extension and implementation details of the algorithms would be another interesting field for future work.

APPENDIX A

PROOF OF THEOREM 3

To prove Theorem 3, we need the following two lemmas for the expected change in degree distribution over one time step.

Lemma 4 *Consider the Random Server algorithm. For all* $t < \min(n, m)$*,*

$$\begin{aligned}
& \mathbb{E}[L_i(t+1) - L_i(t) | L(t), R(t)] \\
= \; & -\frac{L_i(t)}{\sum_j L_j(t)} \frac{R_0(t)}{\sum_j R_j(t)} \\
& + \sum_{k \geq 1} \frac{R_k(t)}{\sum_j R_j(t)} [(k-1)(q_{i+1} - q_i) - q_i] + O\left(\frac{1}{w}\right), \qquad \text{(A.1)} \\
& \mathbb{E}[R_i(t+1) - R_i(t) | L(t), R(t)] \\
= \; & -\frac{R_i(t)}{\sum_j R_j(t)} + \frac{R_0(t)}{\sum_j R_j(t)} \sum_{s \geq 1} \frac{L_s(t)}{\sum_j L_j(t)} s(p_{i+1} - p_i) \\
& + \frac{\sum_{k \geq 1} R_k(t)}{\sum_j R_j(t)} [\sum_{s \geq 1} q_s (s-1)(p_{i+1} - p_i)] + O\left(\frac{1}{w}\right), \qquad \text{(A.2)} \\
& \mathbb{E}[M_p(t+1) - M_p(t) | L(t), R(t)] = 1 - \frac{R_0(t)}{\sum_j R_j(t)}, \qquad \text{(A.3)} \\
& \mathbb{E}[M_q(t+1) - M_q(t) | L(t), R(t)] = \frac{R_0(t)}{\sum_j R_j(t)}, \qquad \text{(A.4)}
\end{aligned}$$

where $q_i = \frac{iL_i(t)}{\sum_j jL_j(t)}$*, and* $p_i = \frac{iR_i(t)}{\sum_j jR_j(t)}$*.*

Lemma 5 *Consider the Peeling algorithm. For all* t *such that* $R_1(t) > 0$*,*

$$\mathbb{E}[L_i(t+1) - L_i(t) | L(t), R(t)] = -\frac{iL_i(t)}{\sum_j jL_j(t)}, i \geq 2, \qquad \text{(A.5)}$$

$$
\begin{aligned}
& \mathbb{E}[R_1(t+1) - R_1(t) | L(t), R(t)] \\
= & \sum_k \frac{kL_k(t)}{\sum_j jL_j(t)}(k-1)(\frac{2R_2(t) - R_1(t)}{\sum_j jR_j(t)}) - 1 + O\left(\frac{1}{w}\right), \qquad \text{(A.6)} \\
& \mathbb{E}[R_i(t+1) - R_i(t) | L(t), R(t)] \\
= & \sum_k \frac{kL_k(t)}{\sum_j jL_j(t)}(k-1)(\frac{(i+1)R_{i+1}(t) - iR_i(t)}{\sum_j jR_j(t)}) + O\left(\frac{1}{w}\right), \\
& \quad for \;\; i \geq 2, \qquad \text{(A.7)} \\
& \mathbb{E}[M_p(t+1) - M_p(t) | L(t), R(t)] = 1, \qquad \text{(A.8)} \\
& \mathbb{E}[M_q(t+1) - M_q(t) | L(t), R(t)] = 0. \qquad \text{(A.9)}
\end{aligned}
$$

Proof of Lemma 4. Consider Eq. (A.1). At step $t+1$, a server node of degree k is selected with probability $\frac{R_k(t)}{\sum_j R_j(t)}$. If $k = 0$, the probability that a task node of degree i is chosen for this server equals $\frac{L_i(t)}{\sum_j L_j(t)}$. This yields the first term in A.1. If $k > 0$, each of these k edges is connected to one of the $\sum_j jL_j(t)$ remaining sockets at the task node sides uniformly at random. This indicates that the edge is connected to a task node of degree i with probability $q_i = \frac{iL_i(t)}{\sum_j jL_j(t)}$. The server node randomly selects one of the k edges, and the connected task node is removed from this graph. In addition, we remove the other $k-1$ edges. If a removed edge is connected to a task node of degree $i+1$, which happens with probability q_{i+1}, the residual degree of this node changes from $i+1$ to i. This explains the factor $(k-1)(q_{i+1} - q_i)$ in the second term of A.1. The extra term $O(1/w)$ comes from the fact that we removed several edges at each time step but we assume that the degree distribution was constant throughout one step.

Similarly, consider Eq. (A.2). If a server node of degree i is selected, it is removed from the graph, yielding the first term in A.2. If a server node of degree 0 is selected, a task node of degree s is selected with probability $\frac{L_s(t)}{\sum_j L_j(t)}$. The task is assigned to this server, and all of its s edges are removed. Each of the s edges connects to a server node of degree i with probability $p_i = \frac{iR_i(t)}{\sum_j jR_j(t)}$. This yields the second term in A.2. If $k > 0$, the probability that a task node of degree s is assigned is q_s. Besides the edge connected to the assigned server, all the other $s-1$ edges are also removed, which leads to the expected change $q_s(s-1)(p_{i+1} - p_i)$ for server node of degree i.

Next consider the evolution of $M_p(t)$ and $M_q(t)$. At step $t+1$, a job is assigned to a q-server only if the selected server has degree 0 , which happens

with probability $\frac{R_0(t)}{\sum_j R_j(t)}$. This gives Eq. (A.4). And the evolution of $M_p(t)$ follows from the fact that $M_p(t+1) + M_q(t+1) = M_p(t) + M_q(t) + 1$. ■

Proof of Lemma 5. At each time step, a server node of degree 1 is selected randomly. The probability that the edge is connected to a degree i task node is $\frac{iL_i(t)}{\sum_j jL_j(t)}$, which yields Eq. (A.5).

Consider Eq. (A.6) and (A.7). With a task node of degree k selected, each of its other $k-1$ edges is connected to a degree i server node with probability $\frac{iR_i(t)}{\sum_j jR_j(t)}$. Removing the edge results in an increase of degree $i-1$ server nodes by 1, and a decrease of server node of degree i by 1. The -1 term in (A.6) comes from the removal of a degree 1 server node at each step. The term $O(1/w)$ again comes from the removal of several edges in one step while assuming the degree distributions remain constant. ■

Proof of Theorem 3. We apply the Wormald method ([46] Theorem C.28). First, we need to construct an open set D that contains the evolution of l_i and γ_i for the Wormald method. Let D be $(-\eta, \frac{1}{d}) \times \Lambda \times \Gamma$, where $Lambda = (0,1)^{l_{max+1}} \backslash \{(l_0, ..., l_{v_{max}}) : \sum l_i \leq \eta\}$, $\Gamma = (0,1)^{r_{max+1}} \backslash \{(\gamma_0, ..., \gamma_{c_{max}}) : \sum \gamma_i \leq \eta\}$. Here, η is strictly positive but arbitrarily small. Then D is an open connected bounded set.

Next, we need to check the following three conditions of the Wormald method in order to assert that typical instances evolve like the sets of differential equations.

(i) (Boundedness) Since at each step, we remove one task node and one server node, we have

$$|L_i(t+1) - L_i(t)| \leq 1, |R_i(t+1) - R_i(t)| \leq 1.$$

(ii) (Trend) Assume that the process evolves like its expected value. We scale the time $\tau = \frac{t}{w}$, where $w = nd$, and let n tend to infinity. Letting $\Delta t = 1$, for the task nodes we have

$$\frac{L_i(t+\Delta t) - L_i(t)}{\Delta t} = \frac{L_i(w\tau + w(d\tau)) - L_i(w\tau)}{w(d\tau)} \approx \frac{dl_i(\tau)}{d\tau}.$$

We can apply the same argument to establish the relationship for $R_i(t)$, $M_p(t)$ and $M_q(t)$ to obtain Eq. (4.1 - 4.8).

(iii) (Lipschitz) Differentiating the right-hand side of Eq. (4.1 - 4.8), we can show that the functions are Lipschitz continuous in $\{l_i, \gamma_j : 1 \leq i \leq$

$l_{max}, 1 \leq j \leq r_{max}\}$.

(iv) (Initial Concentration) At $t = 0$, with n unassigned jobs and k idle servers, the graph ensemble has the fraction $\frac{L_i(0)}{n}$ with a binomial distribution $B(d, \frac{k}{m})$ and $\frac{R_i(0)}{k}$ with a binomial distribution $B(dn, \frac{1}{m})$ truncated at r_{max}. Hence $l_{max} = d$ and we let r_{max} be a fixed constant as there is a limited amount of storage space on each server. The Bernstein inequality yields

$$\mathbb{P}\left(\left|\frac{L_i(0)}{w} - \mathbb{E}\left[\frac{L_i(0)}{w}\right]\right| \geq w^{-\frac{1}{6}}\right) \leq O(e^{-w^{\frac{2}{3}}b}),$$

$$\mathbb{P}\left(\left|\frac{R_i(0)}{w} - \mathbb{E}\left[\frac{R_i(0)}{w}\right]\right| \geq w^{-\frac{1}{6}}\right) \leq O(e^{-w^{\frac{2}{3}}b}),$$

where b is a strictly positive constant. This shows the concentration of the initial condition.

Theorem 3 thus follows from the application of Wormald's method. ■

APPENDIX B

EVOLUTION OF RESIDUAL GRAPH WITH TASKS QUEUEING

Given a random graph ensemble $\mathbf{G}(\mathbf{L}, \mathbf{R})$ for unassigned tasks nodes and idle servers, we have shown the evolution of residual graph for both Random Server algorithm and Peeling algorithm. If there are no outstanding tasks in the system, it is convenient to obtain the fraction of p-servers and q-servers by solving the mean field equations on (L, R). Consider the case of tasks queueing. To apply the mean field model for degree evolution over multiple time intervals, we need the degree distribution of outstanding tasks, as well as that of idle servers, p-servers and q-servers. Hence, for the Random Server algorithm, at every step, we do not remove other edges connected to the selected server except the one connected to the selected task. As for Peeling algorithm, since a server node with only one degree is selected at every step, the procedure is actually the same as described in Section 4.3. Let $\Lambda_{i,j}$ denote the number of tasks with i edges connected to idle servers and j edges connected to p-servers. As the degree of an unassigned task is d, the number of edges connected to q-servers is $d-i-j$. Let Ψ_i, Φ_i and Θ_i denote the number of idle servers, p-servers and q-servers with degree i respectively. Then we can use $G(\Lambda(t), \Psi(t), \Phi(t), \Theta(t))$ to describe the residual graph at step t. We still denote the respective numbers of p-servers and q-servers at the end of time step t. Similarly, define the scaled time $\tau = \frac{t}{w}$, where $w = nd$ denotes the total number of edges in the initial graph. Let $\lambda_{i,j}(\tau) = \frac{\Lambda_{i,j}(w\tau)}{w}$, $\psi_i(\tau) = \frac{\Psi_i(w\tau)}{w}$, $\phi_i(\tau) = \frac{\Phi_i(w\tau)}{w}$, $\theta_i(\tau) = \frac{\Theta_j(w\tau)}{w}$, $m_p(\tau) = \frac{M_p(w\tau)}{w}$, $m_q(\tau) = \frac{M_q(w\tau)}{w}$, which determine the assignment path. Similar to Theorem 1, we have the following theorem

Theorem 8 Evolution of residual graph with occupied servers for the assignment algorithms. *The expected assignment paths of the two algorithms are described by the two sets of differential equations respectively:*

Random Server algorithm:

$$\frac{d\lambda_{i,j}(\tau)}{d\tau} = -\frac{\lambda_{i,j}\phi_0}{v(\tau)c(\tau)} + \sum_{k\geq 1}\frac{\phi_k}{c(\tau)}\frac{(k-1)(i+1)\lambda_{i+1,j-1} - ki\lambda_{i,j}}{e(\tau)},$$

$$for \;\; 0 \leq i+j \leq d, \tag{B.1}$$

$$\frac{d\phi_i(\tau)}{d\tau} = -\frac{\phi_i}{c(\tau)} + \frac{(i+1)\phi_{i+1} - i\phi_i}{\sum_j j\phi_j}[\frac{\phi_0}{c(\tau)}\sum_{0\leq s\leq d}\sum_{0\leq k\leq d-s}\frac{s\lambda_{s,k}}{v(\tau)}$$

$$+(1-\frac{\phi_0}{c(\tau)})\sum_{0\leq s\leq d}\sum_{0\leq k\leq d-s}\frac{s(s-1)\lambda_{s,k}}{e(\tau)}],$$

$$for \;\; 1 \leq i \leq r_{max}-1, \tag{B.2}$$

$$\frac{d\psi_i(\tau)}{d\tau} = \frac{\phi_{i+1}}{c(\tau)} + \frac{(i+1)\psi_{i+1} - i\psi_i}{\sum_b b\psi_b}[\frac{\phi_0}{c(\tau)}\sum_{0\leq s\leq d}\sum_{0\leq k\leq d-s}\frac{s\lambda_{s,k}}{v(\tau)}$$

$$+(1-\frac{\phi_0}{c(\tau)})\sum_{0\leq s\leq d}\sum_{0\leq k\leq d-s}\frac{k\lambda_{s,k}}{e(\tau)}],$$

$$for \;\; 0 \leq i \leq r_{max}-1, \tag{B.3}$$

$$\frac{d\theta_0(\tau)}{d\tau} = \frac{\phi_0}{c(\tau)} + \frac{(i+1)\theta_{i+1} - i\theta_i}{\sum_b b\theta_b}[\frac{\phi_0}{c(\tau)}\sum_{0\leq s\leq d}\sum_{0\leq k\leq d-s}\frac{(d-s-k)\lambda_{s,k}}{v(\tau)}$$

$$+(1-\frac{\phi_0}{c(\tau)})\sum_{0\leq s\leq d}\sum_{0\leq k\leq d-s}\frac{(d-s-k)\lambda_{s,k}}{e(\tau)}], \tag{B.4}$$

$$\frac{d\theta_i(\tau)}{d\tau} = \frac{(i+1)\theta_{i+1} - i\theta_i}{\sum_b b\theta_b}[\frac{\phi_0}{c(\tau)}\sum_{0\leq s\leq d}\sum_{0\leq k\leq d-s}\frac{(d-s-k)\lambda_{s,k}}{v(\tau)}$$

$$+(1-\frac{\phi_0}{c(\tau)})\frac{(d-s-k)\lambda_{s,k}}{e(\tau)}],$$

$$for \;\; 1 \leq i \leq r_{max}-1, \tag{B.5}$$

$$\frac{dm_p(\tau)}{d\tau} = 1 - \frac{\phi_0}{c(\tau)}, \tag{B.6}$$

$$\frac{dm_q(\tau)}{d\tau} = \frac{\phi_0}{c(\tau)}, \tag{B.7}$$

where $v(\tau) = \sum_{0\leq i\leq d}\sum_{0\leq j\leq d-i}\lambda_{i,j}(\tau)$, $c(\tau) = \sum_j \phi_j(\tau)$, *and* $e(\tau) = \sum_{0\leq i\leq d}\sum_{0\leq j\leq d-i} i\lambda_{i,j}(\tau)$.

Peeling algorithm:

$$\frac{d\lambda_{i,j}(\tau)}{d\tau} = -\frac{i\lambda_{i,j}}{e(\tau)} \quad for \;\; 0 \le i+j \le d, \tag{B.8}$$

$$\frac{d\phi_1(\tau)}{d\tau} = -1 + \sum_{0\le s\le d}\sum_{0\le k\le d-s} \frac{s\lambda_{s,k}}{e(\tau)}(s-1)\frac{(i+1)\phi_{i+1} - i\phi_i}{\sum_j \phi_j},$$

$$\frac{d\phi_i(\tau)}{d\tau} = \sum_{0\le s\le d}\sum_{0\le k\le d-s} \frac{s\lambda_{s,k}}{e(\tau)}(s-1)\frac{(i+1)\phi_{i+1} - i\phi_i}{\sum_j \phi_j},$$

$$for \;\; 0 \le i \le r_{max}-1 \;\; and \;\; i \ne 1, \tag{B.9}$$

$$\frac{d\psi_0(\tau)}{d\tau} = -1 + \sum_{0\le s\le d}\sum_{0\le k\le d-s} \frac{s\lambda_{s,k}}{e(\tau)}k\frac{\psi_1}{\sum_j \psi_j},$$

$$\frac{d\psi_i(\tau)}{d\tau} = \sum_{0\le s\le d}\sum_{0\le k\le d-s} \frac{s\lambda_{s,k}}{e(\tau)}k\frac{(i+1)\psi_{i+1} - i\psi_i}{\sum_j \psi_j},$$

$$for \;\; 1 \le i \le r_{max}-1, \tag{B.10}$$

$$\frac{d\theta_i(\tau)}{d\tau} = \sum_{0\le s\le d}\sum_{0\le k\le d-s} \frac{s\lambda_{s,k}}{e(\tau)}(d-s-k)\frac{(i+1)\theta_{i+1} - i\theta_i}{\sum_j \theta_j},$$

$$for \;\; 0 \le i \le r_{max}-1, \tag{B.11}$$

$$m_p(\tau) = \tau \quad and \quad m_q(\tau) = 0 \tag{B.12}$$

To prove Theorem 8, we need the following two lemmas for the expected change in degree distribution over one time step.

Lemma 6 *Consider the Random Server algorithm. For all $t < \min(n, m)$,*

$$\begin{aligned}
&\mathbb{E}[\Lambda_{i,j}(t+1) - \Lambda_{i,j}(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= \; & \sum_{k \geq 1} \frac{\Phi_k(t)}{\sum_j \Phi_j(t)} \frac{(k-1)(i+1)\Lambda_{i+1,j-1}(t) - ki\Lambda_{i,j}(t)}{E(t)} \\
& - \frac{\Lambda_{i,j}(t)}{V(t)} \frac{\Phi_0(t)}{\sum_j \Phi_j(t)} + O\left(\frac{1}{w}\right), \qquad \text{(B.13)}
\end{aligned}$$

$$\begin{aligned}
&\mathbb{E}[\Phi_i(t+1) - \Phi_i(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= \; & -\frac{\Phi_i(t)}{C(t)} + \left(\frac{(i+1)\Phi_{i+1}(t) - i\Phi_i(t)}{X(t)}\right) \\
& [\sum_{k \geq 1} \frac{\Phi_k(t)}{C(t)} \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} (s-1) \frac{s\Lambda_{s,k}(t)}{E(t)} \\
& + \frac{\Phi_0(t)}{C(t)} \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} s \frac{\Lambda_{s,k}(t)}{V(t)}] + O\left(\frac{1}{w}\right), \qquad \text{(B.14)}
\end{aligned}$$

$$\begin{aligned}
&\mathbb{E}[\Psi_i(t+1) - \Psi_i(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= \; & -\frac{\Phi_{i+1}(t)}{C(t)} + \left(\frac{(i+1)\Psi_{i+1}(t) - i\Psi_i(t)}{Y(t)}\right) \\
& [\sum_{k \geq 1} \frac{\Phi_k(t)}{C(t)} \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} k \frac{s\Lambda_{s,k}(t)}{E(t)} \\
& + \frac{\Phi_0(t)}{C(t)} \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} k \frac{\Lambda_{s,k}(t)}{V(t)}] + O\left(\frac{1}{w}\right), \qquad \text{(B.15)}
\end{aligned}$$

$$\begin{aligned}
&\mathbb{E}[\Theta_0(t+1) - \Theta_0(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= \; & \frac{\Theta_1(t)}{Z(t)} [\sum_{k \geq 1} \frac{\Phi_k(t)}{C(t)} \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} \frac{s\Lambda_{s,k}(t)}{E(t)} (d-s-k) \\
& + \frac{\Phi_0(t)}{C(t)} \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} \frac{\Lambda_{s,k}(t)}{V(t)} (d-s-k)] \\
& + \frac{\Phi_0(t)}{C(t)} + O\left(\frac{1}{w}\right), \qquad \text{(B.16)}
\end{aligned}$$

$$
\begin{aligned}
& \mathbb{E}[\Theta_i(t+1) - \Theta_i(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= & \left(\frac{(i+1)\Theta_{i+1}(t) - i\Theta_i(t)}{Z(t)} \right) \\
& [\sum_{k \geq 1} \frac{\Phi_k(t)}{C(t)} \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} (d-s-k) \frac{s\Lambda_{s,k}(t)}{E(t)} \\
& + \frac{\Phi_0(t)}{C(t)} \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} (d-s-k) \frac{\Lambda_{s,k}(t)}{V(t)}] + O\left(\frac{1}{w}\right), \qquad \text{(B.17)}
\end{aligned}
$$

$$
\mathbb{E}[M_p(t+1) - M_p(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] = 1 - \frac{\Phi_0(t)}{X(t)}, \quad \text{(B.18)}
$$

$$
\mathbb{E}[M_q(t+1) - M_q(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] = \frac{\Phi_0(t)}{X(t)}, \qquad \text{(B.19)}
$$

where $V(t) = \sum_{0 \leq i \leq d} \sum_{0 \leq j \leq d-i} \Lambda_{i,j}(t)$, $C(t) = \sum_j \Phi_j(t)$, $E(t) = \sum_{0 \leq i \leq d} \sum_{0 \leq j \leq d-i} i\Lambda_{i,j}(t)$, $X(t) = \sum_j j\Phi_j(t)$, $Y(t) = \sum_j j\Psi_j(t)$ *and* $Z(t) = \sum_j j\Theta_j(t)$.

Lemma 7 *Consider the Peeling algorithm. For all* t *such that* $R_1(t) > 0$,

$$
\mathbb{E}[\Lambda_{i,j}(t+1) - \Lambda_{i,j}(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] = \frac{i\Lambda_{i,j}(t)}{E(t)}, \quad \text{(B.20)}
$$

$$
\begin{aligned}
& \mathbb{E}[\Phi_1(t+1) - \Phi_1(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= & -1 + \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} \frac{s\Lambda_{s,k}}{E(t)} (s-1) \frac{2\Phi_2 - \Phi_1}{X(t)}, \qquad \text{(B.21)}
\end{aligned}
$$

$$
\begin{aligned}
& \mathbb{E}[\Phi_i(t+1) - \Phi_i(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= & \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} \frac{s\Lambda_{s,k}}{E(t)} (s-1) \frac{(i+1)\Phi_{i+1} - i\Phi_i}{X(t)}, \\
& \quad \textit{for} \;\; 0 \leq i \leq r_{max} - 1 \;\; \textit{and} \;\; i \neq 1, \qquad \text{(B.22)}
\end{aligned}
$$

$$
\begin{aligned}
& \mathbb{E}[\Psi_0(t+1) - \Psi_0(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= & -1 + \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} \frac{s\Lambda_{s,k}}{E(t)} k \frac{\Psi_1}{Y(t)}, \qquad \text{(B.23)}
\end{aligned}
$$

$$
\begin{aligned}
& \mathbb{E}[\Psi_i(t+1) - \Psi_i(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= & \sum_{0 \leq s \leq d} \sum_{0 \leq k \leq d-s} \frac{s\Lambda_{s,k}}{E(t)} k \frac{(i+1)\Psi_{i+1} - i\Psi_i}{Y(t)}, \\
& \quad \textit{for} \;\; 1 \leq i \leq r_{max} - 1, \qquad \text{(B.24)}
\end{aligned}
$$

$$
\begin{aligned}
&\mathbb{E}[\Theta_i(t+1) - \Theta_i(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] \\
= &\sum_{0 \le s \le d} \sum_{0 \le k \le d-s} \frac{s\Lambda_{s,k}}{e(\tau)} (d-s-k) \frac{(i+1)\Theta_{i+1} - i\Theta_i}{Z(t)}, \\
&\quad for \;\; 0 \le i \le r_{max} - 1, \qquad \text{(B.25)} \\
&\mathbb{E}[M_p(t+1) - M_p(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] = 1, \qquad \text{(B.26)} \\
&\mathbb{E}[M_q(t+1) - M_q(t) | \Lambda(t), \Phi(t), \Psi(t), \Theta(t)] = 0. \qquad \text{(B.27)}
\end{aligned}
$$

Proof of Lemma 6. Consider Eq. (B.13). At step $t+1$, an idle server with degree k is selected with probability $\frac{\Phi_k(t)}{\sum_j \Phi_j(t)}$. If $k > 0$, each of these k edges is uniformly connected to the $E(t) = \sum_{0 \le i \le d} \sum_{0 \le j \le d-i} i\Lambda_{i,j}(t)$ remaining sockets at the task node sides. Hence the probability that the edge is connected to a task node of degree (i, j) is $\frac{i\Lambda_{i,j}(t)}{E(t)}$. The selected server node randomly chooses one of the k edges, and the connected task node is removed from this graph. As this server becomes a p-server, the other $k-1$ edges become edges connected to p-servers. For any of these $k-1$ edges, if it is connected to a task node of degree $(i+1, j-1)$, which happens with probability $\frac{i\Lambda_{i+1,j-1}(t)}{E(t)}$, the degree of this node changes from $(i+1, j-1)$ to (i, j). This explains the first term in Eq. (B.13). If $k = 0$, a task node with degree (i, j) is selected with probability $\frac{\Lambda_{i,j}(t)}{V(t)}$. This yields the second term in B.13. Similarly, the extra term $O\left(\frac{1}{w}\right)$ comes from the fact that we removed several edges at each step but we asume that the degree distribution was constant throughout one step.

As for Eq. (B.14), it is similar to Eq. (A.2) of original assignment algorithm. We do not repeat here.

Then consider Eq. (B.15). If a server node of degree $i+1$ is selected, $i \ge 0$, it becomes a p-server of degree i, yielding the first term in B.15. Similarly, consider the case that a server node of degree $k > 0$ is selected. Then the probability that a task node of degree (s, k) is chosen for this server is $\frac{s\Lambda_{s,k}(t)}{E(t)}$. We remove all of its connected edges, among which k edges are connected to p-servers. For any of these k edges, with probability $\frac{i\Psi_i(t)}{X(t)}$ it is connected to a p-server of degree i, which changes into degree $i-1$ after we remove the assigned task node. This yields the first term in the square bracket. If $k = 0$, a task node of degree (s, k) is selected with probability $\frac{\Lambda_{s,k}(t)}{V(t)}$, which explains the second term in the square bracket.

Next consider the evolution of q-servers. At step $t+1$, the selection of a server of degree 0 contributes to an increase of q-server of degree 0, which happens with probability $\frac{\Phi_0(t)}{C(t)}$. This yields the third term in Eq. (B.16). Other possible change for the degree of a p-server comes from the remove of task node. This is similar to that of p-server. Note that for a task node of degree (s,k), the number of edges connected to q-servers is $d-s-k$.

The evolution of $M_p(t)$ and $M_q(t)$ is the same as we described in Section (4.3). ■

Proof of Lemma 7. The evolution of $\Lambda(i,j)$ and $\Phi_i(t)$ is similar to that in Lemma (5).

Consider Eq. (B.23). Note that under Peeling algorithm, a server node of degree 1 is selected randomly at each time step. With the connected task node removed, the server becomes a p-server with degree 0, which yields the -1 term in B.23. Similarly, other possible change for the degree of p-server comes from the remove of task node. Since during the pure peeling assignment, the occurrence of q-server is impossible, the only effect for the degree evolution of q-servers results from the remove of task node. Since the explanation for Eq. (B.24) and (B.25) is almost the same as that in the proof of Lemma 6, we do not restate here. ■

Proof of Theorem 8. The proof is similar to that for Theorem 3. With Lemma 6 and 7, we can show that all the four conditions for the application of Wormald method are fulfilled. ■

REFERENCES

[1] M. Armbrust et al., "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, Apr. 2010.

[2] E. Schurman and J. Brutlag, "The user and business impact of server delays, additional bytes and http chunking in web search," in *O'Reilly Velocity Web Performance and Operations Conference*, 2009.

[3] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *Computer*, vol. 40, pp. 103–105, 2007.

[4] "Load balancing 101: Nuts and bolts," white paper, F5 Networks, Inc., July 2007.

[5] G. Ciardo, A. Riska, and E. Smirni, "Equiload: a load balancing policy for clustered web servers," *Performance Evaluation*, vol. 46, pp. 101–124, Oct. 2001.

[6] M. Harchol-Balter, M. Crovella, and C. D. Murta, "On choosing a task assignment policy for a distributed server system," in *Proc. 10th International Conference on Computer Performance Evaluation*, 1998, pp. 231–242.

[7] J. Broberg, Z. Tari, and P. Zeephongsekul, "Task assignment based on prioritising traffic flows," in *Proc. 8th international conference on Principles of Distributed Systems*, 2004, pp. 415–430.

[8] V. Gupta, M. Harchol-Balter, K. Sigman, and W. Whitt, "Analysis of join-the-shortest-queue routing for web server farm," *Performance Evaluation*, vol. 64, pp. 1062–1081, Oct. 2007.

[9] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 45–52, Jan. 2011.

[10] Anon, "Let it rise: a special report on corporate it," *The Economist*, vol. 389, pp. 3–16, Oct. 2008.

[11] N. Ahmad, A. G. Greenberg, P. Lahiri, D. Maltz, P. K. Patel, S. Sengupta, and K. V. Vaid, "Distributed load balancer," U.S. Patent 0 036 903, Aug. 11, 2010.

[12] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. USENIX Symp. Operating Syst. Design and Implementation*, 2004, pp. 137–150.

[13] "Apache hadoop," June 2011. [Online]. Available: http://hadoop.apache.org

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. Eur. Conf. Comput. Syst.*, 2007, pp. 59–72.

[15] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in *Proc. IEEE Intl Conf. Cluster Computing*, 2010, pp. 188–196.

[16] J. Xiong, J. Li, R. Tang, and Y. Hu, "Improving data availability for a cluster file system through replication," in *Proc. IEEE Intl Symp. Parallel and Distrib. Processing*, 2008, pp. 1–8.

[17] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. USENIX Symp. Operating Syst. Design and Implementation*, 2010, pp. 1–7.

[18] M. Tang, B.-S. Lee, X. Tang, and C.-K. Yeo, "The impact of data replication on job scheduling performance in the data grid," *Future Generation Comput. Syst.*, vol. 22, pp. 254–268, 2006.

[19] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.

[20] D. Chappell, "Introducing windows azure," DavidChappell & Associates, white paper, Mar. 2009.

[21] "The gogrid wiki website." [Online]. Available: https://wiki.gogrid.com/wiki/index.php

[22] "The nginx website." [Online]. Available: http://nginx.org/en/

[23] "Haproxy - the reliable, high performance tcp/http load balancer." [Online]. Available: http://haproxy.1wt.eu/

[24] "Amazon web services: Elastic load blancing." [Online]. Available: http://aws.amazon.com/elasticloadbalancing/

[25] "Opennebula: Scaling out web servers to amazon ec2." [Online]. Available: http://opennebula.org/documentation:uc4

[26] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich, "Queueing system with selection of the shortest of two queues: An asymptotic approach," *Probl. Inf. Transm*, vol. 32, pp. 15–27, 1996.

[27] M. Mitzenmacher, "The power of two choices in randomized load balancin," Ph.D. dissertation, Univ. of California Berkeley, 1996.

[28] M. Bramson, Y. Lu, and B. Prabhakar, "Randomized load balancing with general service time distributions," in *ACM Sigmetrics*, 2010, pp. 275–286.

[29] C. Graham, "Chaoticity on path space for a queueing network with selection of the shortest queue among several," *Journal of Appl. Prob*, vol. 37, pp. 198–211, 2000.

[30] M. Luczak and C. McDiarmid, "On the maximum queue length in the supermarket model," *The Annals of Probability*, vol. 34, pp. 493–527, 2006.

[31] M. S. Squillante and R. D. Nelson, "Analysis of task migration in shared-memory multiprocessor scheduling," in *Proc. ACM Conference on the Measurement and Modeling of Computer Systems*, 1991, pp. 143–155.

[32] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 356–368.

[33] M. Mitzenmacher, "Analyses of load stealing models based on differential equations," in *Proc. 10th ACM Symposium on Parallel Algorithms and Architetures*, 1998, pp. 212–221.

[34] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha, "Scalable work stealing," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[35] Y.-T. Wang and R. Morris, "Load sharing in distributed systems," *IEEE Trans. on Computer*, vol. 34, pp. 204–217, Mar. 1985.

[36] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. on Software Engineering*, vol. 12, pp. 662–675, May 1986.

[37] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. Larus, and A. Greenberg, "Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services," in *Performance Evaluation*, 2011, pp. 1056–1071.

[38] M. Bramson, Y. Lu, and B. Prabhakar, "Asymptotic independence of queues under randomized load balancing," *Journal of Queueing Systems: Theory and Applications (QUESTA)*, vol. 71, pp. 247–292, July 2012.

[39] L. Kleinrock, *Queueing Systems: Volume I-Theory.* New York: Wiley Interscience, 1975.

[40] F. Bonomi and A. Kumar, "Adaptive optimal load balancing in a non-homogeneous multiserver system with a central job scheduler," *IEEE Trans. Comput.*, vol. 39, pp. 1232–1250, Oct. 1990.

[41] H.-L. Chen, J. Marden, and A. Wierman, "On the impact of heterogeneity and back-end scheduling in load balancing designs," in *Proc. IEEE INFOCOM'09*, 2009.

[42] Y.-C. Chow and W. H. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput.*, vol. 28, pp. 354–361, May 1979.

[43] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: Coping with skewed popularity content in mapreduce clusters," in *Proc. Eur. Conf. Comput. Syst.*, 2011, pp. 287–300.

[44] C. Abad, Y. Lu, and R. Campbell, "Dare: Adaptive data replication for efficient cluster scheduling," in *IEEE Cluster*, 2011, pp. 159–168.

[45] Q. Xie and Y. Lu, "Degree-guided map-reduce task assignment with data locality constraint," in *IEEE International Symposium on Information Theory*, July 2012.

[46] T. Richardson and R. Urbanke, *Modern Coding Theory.* Cambridge University Press, 2007.

[47] C. Bordenave, M. Lelarge, and J. Salez, "Matchings on innite graphs," *ArXiv preprints*, 2011.